Operating System/2 Programming Tools and Information Version 1.2/1.3

Programming Guide Technical Upgrade

85F1531

Operating System/2 Programming Tools and Information Version 1.2/1.3

Programming Guide Technical Upgrade

First Edition (December 1990)

Special Notices

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
IBM C/2
IBM COBOL/2
IBM FORTRAN/2
IBM Macro Assembler/2
Operating System/2
OS/2
Presentation Manager
Systems Application Architecture.

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Adobe	Adobe Systems Incorporated
Helvetica	Linotype
Microsoft	Microsoft Corporation
PostScript	Adobe Systems Incorporated
Times New Roman	Monotype

About This Book

This book, as well as the other books in the upgrade package, contains programming information that is new to or different from the information in the *IBM* Operating System/2* (OS/2*) Programming Tools and Information Version 1.2.* library (part number 6024929). In addition, information from the *OS/2 Programming Tools and Information Version 1.2 Technical Update* (part number 64F1705) is included in this upgrade package.

The upgrade describes features added by the OS/2 Version 1.3 product, and amends some of the information that was published with OS/2 Version 1.2.

This book is a companion to the *OS/2 Version 1.2 Programming Guide*. **Only those chapters of the Version 1.2 book that are changed by the Version 1.3 product are included herein.** The chapter numbers and titles are the same as those in the Version 1.2 book. As a convenience, the beginning of each chapter in this book provides a summary of changes for that chapter.

Who Should Read This Book

This book is for application designers and programmers who are using the components of the *IBM OS/2 Version 1.2/1.3 Programming Tools and Information Technical Upgrade* package. The reader is assumed to be familiar with the services of OS/2.

* Trademark of the IBM Corporation

Contents

Chapter 3. Windows — Concepts and Fundamentals 1
Limits for Number of Windows 1

Chapter 7. Dialog Boxes and Controls 3
Spin Button 3

Chapter 16. Developing Application Code for the Help Interface 7
Processing Help Requests for a Child Window 7

Chapter 17. Developing Help Information 9
Source File Structure 9
Text Presentation 9

Chapter 18. An Introduction to the Graphics Programming Interface 11
GPI Memory Considerations 11
Micro Presentation Space 11
Normal Presentation Space 13
Fonts 14

Chapter 23. Graphics Text 15
Introduction 15
The Presentation Manager-Supplied Fonts 15
Public and Private Fonts 16
Using Additional Adobe Fonts 16
Anti-Aliased Fonts 16

Chapter 29.1. Direct Manipulation 17
Introduction 17
Using Direct Manipulation in an Application 18
Writing a Source Application 18
Dragging the Objects 21
Completion of a Direct-Manipulation Operation 22
Summary of Direct-Manipulation Functions Used by the Source 23
Writing a Target Application 23
Messages Sent to a Target Application 23
Responding to Messages and Providing Visual Feedback 24
Providing a Customized Mouse Pointer 25
Providing Target Emphasis 25
Keyboard Augmentation 26
Summary of Direct-Manipulation Functions Used by the Target 26
Example - Two Object Drag 28
Application Interaction after a Drop 30
Conversation Initiation 30
Considerations when Establishing a Conversation 30
Determining if Data Can be Exchanged 31
Determining How To Exchange the Data 31
Performance Considerations 31
Using Direct-Manipulation Data Transfer in an Application 32
Example - Conversation after the Drop 33
Standard Rendering Mechanisms 34
OS/2 File 34
Print 36

DDE 36
Application Extensions to the Direct-Manipulation Data-Transfer Protocol 38
Rendering Mechanism Name 38
Native Mechanism Actions 38
Naming Conventions 38
Performance Considerations 39

Chapter 30. Alphanumeric Video Output 41
Setting the Cell Size 41
Setting the Display Mode For Text-Windowed and Full-Screen Applications . . . 41
Display Mode Attributes Support by Adapters (table) 42
Loading Fonts 42

Chapter 32. Memory Management 43
Allocating a Locally Shared Segment 43
Creating and Accessing Discardable Segments 43

Chapter 33. File and Disk Management 45
Controlling Access to EAs 45
Defining EA data structures 46

Chapter 36. Printing 47
Spooler 47
Spooler Calls 47
Existing Applications and Migration 50
Down-Level Support 53
Constants 55
Spooler Return Codes 56

Chapter 39. Supporting National Languages 59
Code Page API Summary 59

Appendix G. System Limits 61
Limits for Various Operating System Resources 61

Appendix H. Information Presentation Facility Tags 63
:hp. (Highlight phrases) 63

Index 65

Chapter 3. Windows — Concepts and Fundamentals

Summary of Changes

New	Updated	Section Title
	√	Limits for Number of Windows

Limits for Number of Windows

On page 3-24 replace the first 3 paragraphs with:

The recommended maximum number of standard windows in a system is 2000.

The recommended maximum number of standard windows in an application is 125.

The above numbers are approximate. The numbers depend on the resources used by each window. The numbers are reduced by extensive use of other resources such as input queues, accelerator tables and window words.

The performance of the system is related to the *total* number of windows in the system. Approximately 125 windows can be allocated from a single storage heap. When the number of windows exceeds 125 (or multiples of 125) a new heap is allocated, causing a slight delay. There is no additional overhead associated with a window being in the second (or higher) storage heap.

The individual components of a standard window are:

Frame
Title Bar (optional)
System Menu (optional)
Action Bar (optional)
Min/Max buttons (optional) (both buttons only mean 1 window)
Client window

Hence most standard windows will contain 6 windows, but it is possible to have fewer windows. To control the contents of standard windows you should specify the FCF_xxxxxx flags explicitly and not rely on FCF_STANDARD.

Dialog boxes consist of frame windows, plus child windows that may be specified in a dialog template. Each button, list box, entry field, or other control defined in the dialog box is a separate child window.

Chapter 7. Dialog Boxes and Controls

Summary of Changes

New	Updated	Section Title
√		Spin Button

Spin Button

When you want your application to give users quick access to a finite set of data, you can use a *spin button*. The spin button allows users to select from a scrollable ring of choices. Since users can see only one item at a time, the spin button control should be used only with data that is intuitively related.

A spin button consists of at least one spin field that is a single line entry field (SLE), and up and down arrows stacked on top of one another positioned at the right of the SLE.

You can create multi-field spin buttons for those applications in which users must select more than one value. For example, in setting a date the spin button control can ask users to set the year, month, and day. The first spin field in the spin button contains a list of years. The second and third spin fields contain a list of months and a list of numbers, respectively.

The application uses a multi-field spin button by creating one master component containing a spin field and the spin arrows and servant components containing only spin fields. At component initialization, the spin buttons are created. The servant components are passed a handle to the master in a message. When a servant spin field has the focus, it is spun by the arrows in the master component.

The list of values in a spin button entry field can be an array of data, or a list of consecutive integers defined by an upper and a lower limit.

Creating the Spin Button Component

A spin button is created as a public window class using WinCreateWindow (WICRT) with a class style of WC_SPINBUTTON and a window style of WS_VISIBLE. These are ORed with any of the spin button style flags. The style flags allow you to specify:

- Character input restrictions (none, numeric, read-only)
- Presentation of the data in the spin field (left-justified, right-justified, centered)
- Presence or absence of a border around the spin field
- Spin speed
- Zero-padding of numeric spin fields.

The placement and width of the spin button component are specified as parameters in WinCreateWindow.

The upper and lower limits of numeric fields, the value array pointer for arrays of strings, and the initial value in the spin field are all set by messages sent from the application to the component.

Destroy the component window using WinDestroyWindow (WIDEL) when finished. The component handle that was returned when the spin button was created is the input parameter to WinDestroyWindow.

User Interaction with the Spin Button

Users can interact with the spin button using either the keyboard or a mouse. Using mouse button 1, users first give focus to the spin field they want changed, and then click on either the Up Arrow or the Down Arrow until the value they want is displayed in the spin field.

Using a keyboard, users press the:

- Up Arrow or Down Arrow key to see the choices
- Left Arrow and Right Arrow keys to move the cursor left and right within a spin field
- Home and End keys to move the cursor to the first and last characters in a spin field
- Tab and Back Tab (Shift+Tab) keys to move the input focus from one field to another in multi-field spin buttons.

Users can view the values in a spin field one at a time, or they can rapidly scroll a list by keeping either arrow pressed. When a spin button is not read-only, users can advance to the value they want to set in a spin field quickly by typing over the value that is currently displayed.

Receiving Spin Button Entry Field Notification Messages

The application receives WM_CONTROL notification messages when the user changes or scrolls the text in the spin entry field.

SPBN_UPARROW	The Up Arrow was clicked on, or the Up Arrow key was pressed.
SPBN_DOWNARROW	The Down Arrow was clicked on, or the Down Arrow key was pressed.
SPBN_SETFOCUS	A spin field was selected and has the focus.
SPBN_KILLFOCUS	A spin field no longer has the focus.
SPBN_ENDSPIN	The user released mouse button 1, or one of the arrow keys, while spinning a button.
SPBN_CHANGE	The contents of the spin field have changed.

Sending Spin Button Entry Field Control Messages

These messages are sent by the application to the component.

SPBM_OVERRIDESETLIMITS	Set or reset the limits in a numeric field.
SPBM_QUERYLIMITS	Query the limits of a numeric field.
SPBM_QUERYVALUE	Show the current value of the spin field.
SPBM_SETARRAY	Set or reset the array of data.

SPBM_SETCURRENTVALUE	Set or reset the current numeric value or array index.
SPBM_SETLIMITS	Set or reset the limits of a numeric field.
SPBM_SETMASTER	Identify the master of a servant component.
SPBM_SETTEXTLIMIT	Set the maximum number of characters allowed in a spin field. The default is 255.
SPBM_SPINDOWN	Show the previous field (spin backward).
SPBM_SPINUP	Show the next field (spin forward).

Chapter 16. Developing Application Code for the Help Interface

Summary of Changes

New	Updated	Section Title
	√	Processing Help Requests for a Child Window

Processing Help Requests for a Child Window

On page 16-13, after the last paragraph, add the following:

In Presentation Manager, parent-child windows are active at the same time. Therefore, when a help instance is associated with a window, its descendents are included in the association. However, only the parent window is the active *help* window. This is because IPF filters help requests through the help subtable associated with the parent window.

For IPF to process help requests for a child window, an application must send IPF an HM_SET_ACTIVE_WINDOW message to set the child window as the active help window. Until this happens, IPF will continue to satisfy help requests for the child window from the helpsubtable for the parent window.

The application, in this case, the child window procedure, may send an HM_SET_ACTIVE_WINDOW message to IPF in response to one of the following messages:

WM_INITMENU: If a user selects a choice from the application action bar, the window procedure receives the message WM_INITMENU.

WM_HELP: If a user presses the F1 key, the window procedure for the window that has the keyboard focus, receives the message WM_HELP. When a child window is created, it automatically has the focus. However, the user can request help for many items on the child window while it has the focus. Because of this, the window procedure needs to send IPF the message HM_SET_ACTIVE_WINDOW, only in response to the next message.

WM_SETFOCUS: When there is no application action bar, or the child window does not have the focus, the application must shift the keyboard focus to the child window. An application can change the keyboard focus by calling WinSetFocus. This results in Presentation Manager sending the message WM_SETFOCUS to the window procedure of the new focus window.

When the keyboard focus shifts from the child back to the parent, the parent window procedure must send IPF an HM_SET_ACTIVE_WINDOW message to set the parent window as the active help window. The parent window procedure then has message processing similar to the child window.

Chapter 17. Developing Help Information

Summary of Changes

New	Updated	Section Title
	√	Source File Structure
	√	Text Presentation

Source File Structure

On page 17-5, after the last paragraph, add the following section:

Source File Requirements

During the compilation process, IPFC creates files to hold data temporarily and then erases them when it no longer needs them. The names of these files are:

filename.mdf
filename.clf
0000
2222

where:

filename
is the name of your source file.

Do not give your source file an extension of .MDF or .CLF. Also, do not give your source file a name of 0000 or 2222.

Text Presentation

On page 17-6, after the first paragraph, add the following:

Note: The maximum size of a line of text processed by IPF is 255 characters.

Chapter 18. An Introduction to the Graphics Programming Interface

Summary of Changes

New	Updated	Section Title
√		GPI Memory Considerations

GPI Memory Considerations

On page 18-2, after the last paragraph, add the following section:

Micro Presentation Space

Although run-time memory requirements vary according to the graphics function an application uses, a 'typical' Micro Presentation Space (MicroPS) graphical application uses 315KB of GPI, engine and display memory resource (code and data).

To reduce the memory required, the GPI MicroPS calls have been packaged into two segments, depending on their frequency of use.

The frequently-used calls segment requires 11KB of memory (included in the 315KB quoted above). If a call is referenced in the second MicroPS segment, this segment is loaded increasing the memory required to run the application by 14KB to 329KB.

The calls in the frequently-used segment are:

Attribute calls

GpiMove
GpiQueryAttrs
GpiQueryBackColor
GpiQueryBackMix
GpiQueryCharBox
GpiQueryCharSet
GpiQueryColor
GpiQueryCurrentPosition
GpiQueryLineType
GpiQueryLineWidth
GpiQueryMix
GpiQueryPatternSet
GpiSetArcParams
GpiSetAttrs
GpiSetBackColor
GpiSetBackMix
GpiSetCharBox
GpiSetCharSet
GpiSetColor
GpiSetCurrentPosition
GpiSetLineType
GpiSetLineWidth

GpiSetMix
GpiSetPattern
GpiSetPatternRefPoint
GpiSetPatternSet

Bit map calls

GpiCreateBitmap
GpiDeleteBitmap
GpiQueryBitmapBits
GpiQueryBitmapHandle
GpiQueryBitmapParameters
GpiQueryDeviceBitmapFormats
GpiSetBitmap
GpiSetBitmapId

Device calls

DevCloseDC
DevEscape
DevOpenDC
DevPostDeviceModes
DevQueryCaps
DevQueryDeviceNames
DevQueryHardcopyCaps

Drawing calls

GpiBeginArea
GpiBeginPath
GpiBitBlt
GpiBox
GpiCharString
GpiCharStringAt
GpiCharStringPosAt
GpiEndArea
GpiEndPath
GpiErase
GpiFillPath
GpiIntersectClipRectangle
GpiLine
GpiPaintRegion
GpiPartialArc
GpiPolyLine
GpiPolySpline
GpiQueryCharStringPos
GpiQueryClipBox
GpiQueryPel
GpiQueryTextBox
GpiRectVisible
GpiRestorePS
GpiSavePS
GpiSetClipRegion
GpiSetPel
GpiStrokePath
GpiWCBitBlt

Font calls

GpiCreateLogFont
GpiDeleteSetId
GpiQueryFontMetrics
GpiQueryFonts
GpiQueryNumberSetIds
GpiQueryWidthTable
GpiSetCP

Font Load calls

GpiLoadFonts
GpiUnloadFonts

Logical Color Table calls

GpiCreateLogColorTable
GpiQueryColorIndex
GpiQueryNearestColor
GpiQueryRealColors
GpiQueryRGBColor

Presentation Space calls

GpiCreatePS
GpiDestroyPS
GpiQueryDevice
GpiResetPS
GpiSetPS

Region calls

GpiCombineRegion
GpiCreateRegion
GpiDestroyRegion
GpiQueryRegionBox
GpiRectInRegion
GpiSetRegion

Transform calls

GpiConvert
GpiQueryModelTransformMatrix
GpiSetModelTransformMatrix
GpiSetPageViewport

Normal Presentation Space

If a normal Presentation Space is used (for example, if *retain* mode is used), or if metafiling is carried out, the run-time memory usage is greater. A normal Presentation Space requires 114KB more than a MicroPS; using *retain* mode adds a further 46KB (that is, a total of 475KB)

Retain (or 'store') mode is used when the application needs to store graphics - typically from user input. Once stored in memory, the graphics can be redrawn or edited as required. As well as the code and data requirements, each graphics order needs approximately 11 bytes of storage. Therefore, an application which uses 2000 graphics drawing orders will need around 22KB of memory to store them.

Fonts

When developing an application which requires the use of fonts, the overhead of using outline fonts should be considered. Outline fonts provide more flexibility, but may use more memory than bit-map fonts.

Using outline (Adobe* Type 1) fonts requires 90KB of memory, the space for the font outline definition (typically 10-30KB), plus the cache space for the generated bit maps. Bit-map fonts require little more room than the font itself. However, the bit-map font contains 330 characters, whereas the outline cache contains just the characters needed, typically only 70. Therefore for larger size fonts, this smaller number of characters in memory will more than compensate for the additional code and font outline definition.

The decision to use outline fonts is dependent on the application being developed. A word processor or graphics application will almost certainly require outline characters to be functionally adequate for the user; however, bit-map fonts may be sufficient for an editing program.

Chapter 23. Graphics Text

Summary of Changes

New	Updated	Section Title
	√	Introduction
	√	The Presentation Manager-Supplied Fonts
	√	Public and Private Fonts
√		Anti-Aliased Fonts

Introduction

On page 23-1, in the third paragraph, delete the sentence that begins 'Outline characters can be...'.

The Presentation Manager-Supplied Fonts

On page 23-1, in the final paragraph of this section, delete the second sentence ('In addition, each font...'). Change the final sentence in the same paragraph to read '... image fonts (in the small font sizes) may have a better appearance...'.

Add the following new information:

The outline fonts now provided by Presentation Manager are Adobe Type 1 fonts. The appearance and performance of these fonts approaches those obtained from image fonts.

The set of fonts now provided by Presentation Manager is:

Times New Roman*:

Times New Roman
Times New Roman Bold
Times New Roman Bold Italic
Times New Roman Italic.

Helvetica*:

Helvetica
Helvetica Bold
Helvetica Bold Italic
Helvetica Italic.

Courier:

Courier
Courier Bold
Courier Bold Italic
Courier Italic.

Symbol:

Symbol.

The old font names (for example, Helv, Tms Rmn) are still supported, but the corresponding new fonts are obtained if these names are selected.

Over 600 Adobe Type 1 fonts are available from font vendors, and users may have a large number of these installed. Applications should therefore be capable of working correctly in these conditions.

These fonts can be used in the same way as other outline fonts. They also contain kerning-pair information (for details about kerned fonts, see "Proportional Spacing and Kerning" on page 23-7). Note that the fonts are more stylized than previously, and contain non-zero a-space and c-space data for each character. These values may be negative (for example, italic characters have negative c-space). The GpiQueryTextBox call reflects these values (for example, the concatenation-point x-value may be to the left of the top-right and bottom-right x-values for italics).

Public and Private Fonts

On page 23-2, in the second paragraph, delete the sentence beginning 'To retrieve the names of all font files...'.

Add the following new information to the end of this section:

Using Additional Adobe Fonts

Adobe Type 1 fonts provided with an application for use by that application only can be loaded using a GpiLoadFonts (GSFLO) call that obeys the following rules:

- The .AFM file of the font must be specified as the font file.
- The .PFB file must be in the same directory as the .AFM file.

Anti-Aliased Fonts

On page 23-2, add the following new topic:

Anti-aliased fonts are available for use with the XGA Device Drivers. The XGA Device Drivers and fonts are supplied with the XGA hardware.

Further information can be found on *IBM Personal System/2 XGA Device Drivers, Diskette 2 - OS/2 1.2 support*, document number 64F5117, which accompanies the XGA hardware. The file READ.ME contains an introduction to the fonts and general information on XGA Device Drivers. The file AAFONTS.TXT contains programming guidance on using the fonts in applications.

Chapter 29.1. Direct Manipulation

Summary of Changes

New	Updated	Section Title
√		Direct Manipulation

Insert this new chapter after **Chapter 29.**

Introduction

Direct manipulation is a protocol that enables the user to visually drag an object in a window and drop it on another object. This causes an interaction, or data exchange, between the window with the object being dragged and the window with the object being dropped on. The window containing the dragged object is referred to as the *source*. The window containing the object that was dropped on is referred to as the *target*. The source and the target may be the same window, different windows within the same application, or windows belonging to different applications. The dragged object may be the only visual object in the source window, or it may be one of many objects. The target object may be the only visual object in the target window, or it may be one of many objects. A source or target that contains multiple objects is referred to as a *container window*.

The data exchange, or interaction, that occurs between the source and target after a direct-manipulation operation enables applications in the system that support the protocol to easily integrate, while providing a simple interface to the user.

A sample program DRAGDROP, is provided with the Toolkit to demonstrate the use of the direct-manipulation protocol. DRAGDROP is a simple directory-navigation application. Two copies of this sample must be running to drag something from one window to the another.

When you start DRAGDROP, the contents of the root directory on drive C: are displayed in a list. Directories are displayed with a leading "/" character.

You can navigate downward in the directory tree by double clicking on a directory displayed in the list. You also can navigate downward by selecting **File** on the Action Bar then **Open....** from the File pull-down. This dialog also can be used to navigate upward in the directory, or to select another drive.

To move or copy files or directories from one directory window to another, use button 1 to select one or more files or directories in the source application. Then press and hold Button 2 and drag the objects to the target application. Release Button 2 when the pointer is over your intended target location. The selected files will then be moved or copied from the source directory to the target directory; the contents of the windows are updated automatically. The default operation is a Move, but you can change this to a Copy by pressing the Ctrl key along with Button 2 of the mouse.

Using Direct Manipulation in an Application

The application responsibilities during a direct-manipulation operation vary, depending on whether the application's window procedure is acting as the source or as the target of the operation. This section describes the responsibilities of the source and target.

Writing a Source Application

The source is responsible for starting a direct manipulation operation. The operation only can be accomplished with the mouse. The operation starts when the application detects that a mouse button has been pressed and the mouse has moved. The dragging continues until terminated, usually when the mouse button is released.

Preparing for the Drag

The direct-manipulation protocol enables the application to use any mouse button for dragging. It is recommended that mouse button 2 be used for direct-manipulation operations.

There is an example "Example - Two Object Drag" on page 28 that shows the sequence of API function and message flow for a typical direct-manipulation operation. The flow shows a two-object drag from App1 to App3, dragging over App2 before reaching App3.

The direct-manipulation operation was started by the source window procedure after the user selected the object or objects to be manipulated and the source detected a WM_BUTTON2DOWN followed by WM_MOUSEMOVE.

Steps to Prepare for the Drag: The source has several responsibilities in preparing for the actual drag of the object or objects across the screen. It must:

- Allocate and initialize the DRAGINFO structure that will convey the necessary information about each object to the target.
- Initialize a set of DRAGIMAGE structures that describe the image to be displayed during the drag operation.
- Make the type of each object being directly manipulated known to the system.
- Make the rendering mechanism and format for each object known to the system. For detailed information see "Standard Rendering Mechanisms" on page 34.
- Make the suggested name of the object at the target known to the system.
- Make the name of the container or folder containing the source object known to the system.
- Make the name of the object at the source known to the system.
- Make the "true" type of each object being directly manipulated known to the system.
- Make the native rendering mechanism and format for each object known to the system.

To prepare for the drag operation, the source must invoke DrgAllocDraginfo to allocate the memory to be used for the DRAGINFO structure. DrgAllocDraginfo will initialize the DRAGINFO structure as follows:

- **cbDraginfo:** The size, in bytes, of the entire DRAGINFO structure, including the DRAGITEM array.
- **cbDragitem:** The size, in bytes, of each DRAGITEM structure.
- **usOperation:** Initialized to DO_DEFAULT.
- **xDrop** and **yDrop:** Initialized to the current mouse-pointer location, in desktop coordinates.
- **cditem:** Initialized to the count of objects being dragged, as specified in DrgAllocDraginfo.

The source is responsible for initializing the **hwndSource** field in the DRAGINFO structure.

The source then completes the initialization of each DRAGITEM as appropriate for each of the objects being dragged. This can be accomplished by using the DrgSetDragitem function, or by obtaining a pointer to each DRAGITEM structure with DrgQueryDragitemPtr and initializing it directly.

The first step that the source takes to initialize the DRAGITEM structure is to create the appropriate drag string handles. String handles must be created for:

- The type or types for the object
- The rendering mechanisms and formats supported for the object
- The suggested name of the object at the target
- The name of the container containing the object (if the object is contained in a container or folder)
- The name of the object at the source when the source will allow the target to carry out the operation for the object.

The balance of the DRAGINFO structure can then be initialized as appropriate for that object.

Type: To directly manipulate an object, both the source and target must know the object type, and understand that type. The **hstrType** field in the DRAGITEM structure conveys this information for each object being dragged. The type is represented by a string handle. The target should check to see if it understands the type prior to allowing the user to drop the object.

Several DTYP_* constants are defined as notational conveniences for common types of data. An application can extend these types by defining its own character strings and creating string handles for them using the DrgAddStrHandle function.

True Type: The true type of an object is the type that most accurately describes the object. For example, the input to a C compiler could have the type "Plain Text" (DRT_TEXT), but would be more accurately described as "C Code" (DRT_C). "C Code" would be the true type of this object.

Multiple types can be conveyed by using a comma to separate strings, as follows:

```
type,type
```

The true type should appear first in the list of types, so the type string for the example object would be "C Code, Plain Text."

Rendering Mechanism and Format: The rendering mechanism and format is a string handle. The string takes the form:

```
"elem [,elem,elem...]"
```

where *elem* is an ordered pair in the form:

```
"<mechanism,format>"
```

or a cross product in the form:

```
"(mechanism[,mechanism...]) X (format[,format...])"
```

Multiple cross products are permitted in a single rendering mechanism and format string handle, as are combinations of ordered pairs and cross products. When cross-product notation is used, the rendering mechanism is the left operand. When ordered-pair notation is used, the rendering mechanism is the left element in the ordered pair.

The rendering mechanism represents the way in which you wish to exchange the data, for example *DDE*. The rendering format identifies the actual format of the data, for example *text*. To exchange data, both the source and target must know how to communicate with each other through the rendering mechanism and understand the particular format of the data. The target should verify that it understands the rendering mechanism and format before allowing the user to drop the object or objects. The rendering mechanism and format are passed as a string handle in the DRAGITEM structure. The string handle must be created using the DrgAddStrHandle function.

Several constants are defined for common rendering mechanisms and formats. An application can extend these by defining its own "<*mechanism,format*>" strings and creating string handles for these using the DrgAddStrHandle function.

For example, if an application understands and can generate a LU 6.2 data stream, it could define its own rendering format, "DRF_LU62," and use it in direct-manipulation operations. If an application wishes to use its own rendering mechanisms or formats to communicate with other applications, it should publish the protocol for the mechanisms, the format of the data streams, or both.

Native Rendering Mechanism and Format: The native rendering mechanism and format of the object is the mechanism that most-naturally conveys the data and the current format of the data. For example, the native rendering mechanism and format for:

- A C source file might be <DRM_OS2FILE,CF_OEMTEXT>
- A spreadsheet file might be <DRM_OS2FILE,CF_SYLK>

It may be possible in some direct-manipulation operations, for the target to carry out the necessary action on the source object without the source's participation. However, this may be possible only when the target understands both the true type and the native rendering mechanism and format of the object. Even when the target will not be carrying out the necessary action on the source object, it is still important to know the native rendering mechanism and format. In making a determination of the rendering mechanism and format to be used in the data exchange after the drop, the target might select the native one, as performance generally will be better when the native rendering mechanism and format is used.

The native rendering mechanism and format is conveyed to the target by making it the first ordered pair, or the first ordered pair to result from a cross product, in the list of rendering mechanism and formats passed in the DRAGINFO structure.

Suggested Name at Target: When dragging an object, for example a file, from one container to another, it is important to know the name that the object should have at the target. This may or may not be the same name it had at the source. This name enables the target to check if another object with the same name already exists at the target, and take the appropriate action. For example, a target container may not allow the user to drop the object or objects if an object by that name already exists at the target.

Container Name: It sometimes may be necessary for a target container to be aware of the name of the source container. This name may carry some location information. For example, the default operation when dragging objects between containers is a Move operation. However, in the case of file folders on different drives, this default would be changed to a Copy operation. Thus, a file folder would fill this field with the drive and path information for the file; for example, (A:\SUBDIR1\SUBDIR2\). A database container, on the other hand, might fill this field with the fully qualified OS/2 file name of the database.

Source Name: It may be possible in some direct-manipulation operations, for the target to carry out the necessary action on the source object without the source's participation. If the source is willing to allow this, when possible, then the source name should be filled in with the name of the source object. For example, a file folder would put the name of the source file in this field, for example, (autoexec.bat). A database manager, on the other hand, might fill this field with some necessary location information so that the target could locate a particular record, or field, within the database.

Dragging the Objects

Once initialization is complete, the source invokes the DrgDrag function to accomplish the direct-manipulation operation. As the mouse moves around the screen, the system will send a DM_DRAGOVER message. The window that receives the DM_DRAGOVER message (the target) would respond with DOR_DROP if it understood the type and rendering formats of the objects being dragged, as well as the operation being performed. When a potential target cannot allow the objects to be dropped at this location in its window, it should respond with DOR_NODROP or DOR_NODROPOP. When a potential target cannot allow the objects to be dropped anywhere in its window, it should respond with DOR_NEVERDROP. This last case prevents multiple DM_DRAGOVER messages being unnecessarily sent to a window when another mouse movement occurs, or when the user presses another augmentation key.

To determine the proper reply to a DM_DRAGOVER message, the target obtains information about the direct-manipulation operation by using the DrgAccessDraginfo function. The DM_DRAGOVER message contains a pointer to the DRAGINFO structure. The target can access this structure with the DrgAccessDraginfo call. This makes all information about the direct-manipulation operation available to the target window.

If the target responds with DOR_NODROP to DM_DRAGOVER, the system changes the image displayed to indicate a drop is not allowed. When the user moves the mouse, or presses, or releases an augmentation key, the system sends another DM_DRAGOVER message.

If a DM_DRAGOVER message receives a reply of DOR_NODROPOP, the system changes the image displayed to indicate a drop is prohibited until the user moves the mouse outside the current target window or presses another augmentation key. When one of these events occurs, DrgDrag sends a DM_DRAGOVER message again. If the user presses another augmentation key but has not moved the mouse, a DM_DRAGOVER message will be sent to the same window, giving it an opportunity to accept the drop for the new operation.

If DOR_NEVERDROP is returned from DM_DRAGOVER, further DM_DRAGOVER messages will not be sent to the target until the mouse pointer is moved outside of and back into the target window. A no-drop image will be displayed.

Application-Defined Drag Operations

This protocol defines a method for integrating two unrelated applications through direct manipulation. At times it may be useful for an application to define its own drag operation to facilitate functions between two windows in the same application or between closely related applications. For example, an application implementing a keyboard remapping function may want to provide a method of redefining keys with direct manipulation. This application could define its own operation, KEYSWAP, whereby dragging one key to another exchanges the definitions of the two keys. This protocol provides the extendibility to enable this kind of function.

Completion of a Direct-Manipulation Operation

The user can end a direct-manipulation operation in one of three ways:

- By pressing the **Esc** key to cancel the operation.
- By releasing the mouse button when the pointer is over a target that cannot accept the drop, this action is equivalent to pressing the **Esc** key. When the pointer is over a target that can accept the drop, the target is informed of the drop, and the source is given the window handle of the target.
- By pressing F1 to request help, a DM_DROPHELP message is posted to the target. This enables the target to provide the user with assistance regarding:
  - What would happen if the user dropped the object or objects on that target.
  - Why the target cannot accept a particular drop.

  The source sees this termination of the direct-manipulation operation as a cancelation.

Dropping the Objects

When the user drops the objects, a WM_BUTTON2UP message is posted to the source. A DM_DROP message is sent to the target, providing it with the information necessary to process the objects that were dropped. The target application uses the information provided to exchange data with the source. The protocol to be used depends upon the rendering mechanism specified for each object. It is the responsibility of the target to establish the appropriate conversation or conversations. It is the responsibility of the source to cooperate in the establishment of the necessary conversation or conversations to achieve the actual data exchange. After completing the direct-manipulation operation, including the post-drop conversation with the source, the target uses DrgDeleteStrHandle or DrgDeleteDraginfoStrHandle to delete the string handles in the DRAGINFO structure, and DrgFreeDraginfo to release the storage.

Summary of Direct-Manipulation Functions Used by the Source

Function	Description
DrgAllocDraginfo	Allocates a DRAGINFO structure in shared memory.
DrgAddStrHandle	Creates a handle for an input string.
DrgDrag	Handles movement of the source-specified pointer around the screen. Provides visual feedback to the user.
DrgFreeDraginfo	Deallocates the memory associated with a DRAGINFO structure.
DrgSetDragitem	Initializes each object element in a DRAGINFO structure

Writing a Target Application

The target in a direct-manipulation operation is responsible for determining whether a particular set of objects can be dropped on it, and aids in providing the user with visual cues regarding the operation. A target is informed of the operation through messages that are sent to it as the pointer provided by the source is dragged across the screen.

When a set of objects are dropped on the target, the target is responsible for establishing the appropriate conversation or conversations with the source to accomplish the data transfer. The type of conversation for each object will be based on the rendering mechanism and format of the object being dropped.

Messages Sent to a Target Application

The following messages are sent to each window whose boundaries are crossed as the user drags the object or objects about the screen.

Message	Description
DM_DRAGOVER	Sent to the window under the pointer as the pointer is dragged across it. A single DM_DRAGOVER message is sent each time the mouse moves and each time a key is pressed or released, and contains a pointer to the DRAGINFO structure. The target can access this structure with the DrgAccessDraginfo function.
DM_DRAGLEAVE	Sent whenever DM_DRAGOVER is sent to a window, and the mouse pointer is moved outside the bounds of that window. If the target or an object in the window had been emphasized as a target, it should be de-emphasized. **Notes:** 1. Container windows should monitor the position of the pointer on DM_DRAGOVER messages and simulate DM_DRAGLEAVE when the pointer moves onto or off of a contained object. 2. A DM_DRAGLEAVE message is not sent if the user drops the objects being dragged within the window. Therefore, when DM_DROP is received, the application de-emphasizes any target that had been emphasized as a valid target.

DM_DROP	Sent to the target to provide it with the information necessary to establish a conversation for data exchange with the source. The target should immediately remove any target emphasis. The data transfers must not be done before responding to the DM_DROP message.
DM_DROPHELP	Posted to a target to indicate that the user requested help for the drag operation while over that target.

Responding to Messages and Providing Visual Feedback

DM_DRAGOVER: This message is sent to a target whenever the user drags the mouse pointer into the window. To assess whether a drop can be accepted, the target must use the DrgAccessDraginfo function to get access to the DRAGINFO structure. It then makes a determination as to whether a drop can be accepted for each object. Several factors come into play in making this determination, including the following:

- Both the source and target must support at least one common rendering mechanism and format.
- The target must understand at least one of the data types for the object.

There are four possible responses available to the target.

Response	**Meaning**
DOR_DROP	The target should send DOR_DROP in response to DM_DRAGOVER if the objects being dragged are acceptable. The drop will not occur unless DOR_DROP is returned.
DOR_NODROP	The target should send DOR_NODROP if the objects being dragged are acceptable, and the current operation is supported by the target, but the objects cannot be dropped in the current location in the target window. For example, a list box might return DOR_NODROP if it contains objects that can be dropped on, but the mouse is over an object that cannot be dropped on. If the target response is DOR_NODROP, DM_DRAGOVER will continue to be sent to it when: • A mouse movement occurs. • A keyboard key is pressed. • The mouse is moved out of and back into the window.
DOR_NODROPOP	The target should send DOR_NODROPOP if it can accept the objects being dragged, but does not support the current operation. This response implies that the drop may be valid if the drag operation changes. Once the target has sent DOR_NODROPOP, no further DM_DRAGOVER messages will be sent to it until: • A keyboard key is pressed • The mouse is moved out of and back into the window.
DOR_NEVERDROP	The target should use this response when it will never accept the objects being dragged. Once the target has responded with DOR_NEVERDROP to a DM_DRAGOVER message, no further DM_DRAGOVER messages will be sent to that target until the user has dragged outside of and back into the target window.

If a reply other than DOR_DROP is received from a target, the augmentation emphasis is automatically changed to indicate that no drop is allowed. This gives the user a visual cue that a drop cannot occur. The emphasis will revert to "drop allowed" when a DOR_DROP reply has been received from some target.

Providing a Customized Mouse Pointer

The target can provide a customized pointer to be displayed while it is the target of the drop. This is accomplished by calling DrgSetDragPointer before responding to the DM_DRAGOVER message. This capability may be used by a target to provide additional visual feedback to the user. The mouse pointer will revert to the default when the pointer is moved to a new target.

Providing Target Emphasis

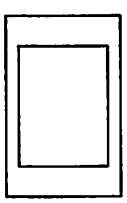
The target should provide target emphasis so the user knows exactly where the drop will occur, or if the drop is not allowed, knows the boundaries of the region where the drop is not allowed.

If the user drags the mouse outside of the target window, resulting in a new target, a DM_DRAGLEAVE message is sent to the former target. The receiver of a DM_DRAGLEAVE message should use this to de-emphasize the target to provide the user with visual feedback that it is no longer the target.

A container window should emphasize a target object by drawing a thin black rectangle around it. The application should use DrgGetPS and DrgReleasePS to obtain the presentation space in which to draw target emphasis.

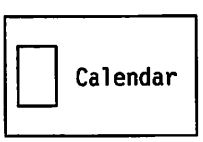
For example, if the object is represented by an icon, it would appear as shown in the following figures. Other forms of target emphasis may be appropriate, depending on the application. CUA is the determining authority for the user interface. Consult your CUA documentation for additional guidelines.

Target emphasis in icon view - emphasis is in icon only:



Calendar

Target emphasis in name view - emphasis is on icon and text:



Target emphasis in text view - emphasis is on text:

Keyboard Augmentation

A direct-manipulation operation begins in a default state. That is, when the user drops the object or objects on a target, the target is informed that it should perform its default operation. It is the target's responsibility to define its default operation. For a container window, the default should be a Move operation, if it is supported. The default for a device, such as a printer, should be a Copy operation.

As the user drags the object or objects, the default operation can be overridden by pressing and holding one of the following augmentation keys.

- **Ctrl:** Changes the operation to a Copy.
- **Alt:** Changes the operation to a Move.

The last key pressed and held at the time of the drop determines the operation to be performed. The target can determine the defined augmentation key that was pressed at the time of the drop by inspecting the **usOperation** member of the DRAGINFO structure.

A target can define additional augmentation keys for its own use. In this case, **usOperation** would indicate that the operation was unknown, and the target would need to use the WinGetKeyState function to determine the actual augmentation key that was used.

Operation Emphasis: As the user presses augmentation keys, the mouse pointer currently being displayed will be modified to provide the user with a visual cue as to the type of operation being performed. sent to the .*target, the target must delete the drag

Summary of Direct-Manipulation Functions Used by the Target

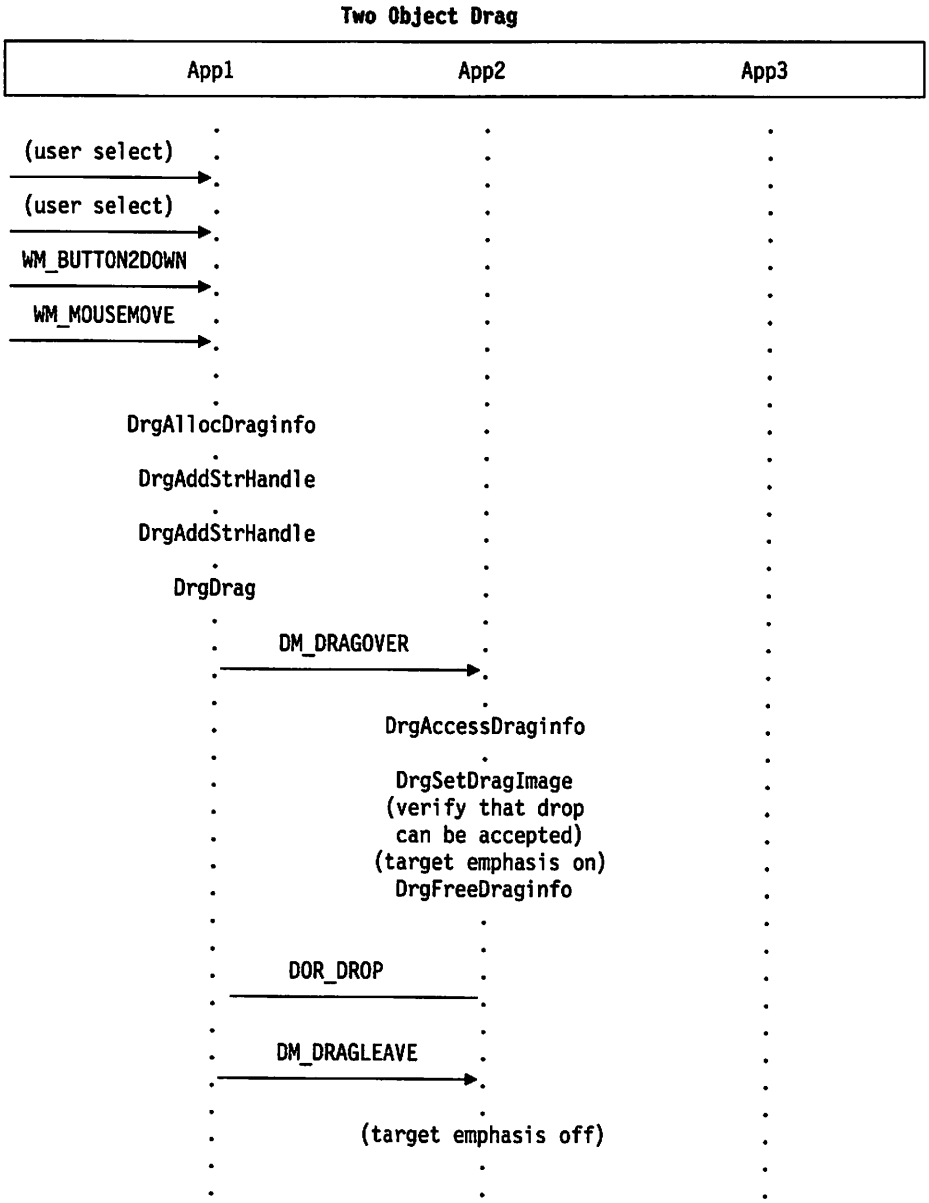
Function	Description
DrgAccessDraginfo	Provides access to the shared segment containing the DRAGINFO structure.
DrgDeleteStrHandle	Disassociates a string from the handle that was assigned to it by DrgAddStrHandle.
DrgDeleteDraginfoStrHandles	Does a DrgDeleteStrHandle for all string handles in a DRAGINFO structure.
DrgFreeDraginfo	Releases the memory associated with a DRAGINFO structure. This function should be called when the target no longer has need for the DRAGINFO structure, or had previously called DrgAccessDraginfo, or a drop had occurred.
DrgGetPS	Unlocks the screen and returns a handle to a cache presentation space that the target can use to provide target emphasis.
DrgQueryDragitem	Copies a given object in a DRAGINFO structure.
DrgQueryDragitemCount	Returns the number of objects involved in a drag operation.
DrgQueryDragitemPtr	Returns a pointer to a given DRAGITEM structure.
DrgQueryNativeRMF	Returns the ordered pair representing the native rendering mechanism and format for an object.

DrgQueryNativeRMFLen	Returns the length of the string representing the native rendering mechanism and format of an object, excluding the null terminating byte.
DrgQueryStrName	Returns the contents of a string associated with a given string handle that was created by DrgAddStrHandle.
DrgQueryStrNameLen	Returns the length of the string associated with a given string handle that was created by DrgAddStrHandle.
DrgQueryTrueType	Returns the string representing the true type of an object being dragged.
DrgQueryTrueTypeLen	Returns the length of the string representing the true type of an object being dragged, *excluding* the null terminating byte.
DrgReleasePS	Releases the cache presentation space obtained using the DrgGetPs function.
DrgSetDragImage	Enables a target to provide a customized image to be dragged.
DrgSetDragPointer	Enables a target to provide a customized mouse pointer while it is the target of a drop.
DrgVerifyNativeRMF	Verifies if the native rendering mechanism and format for a particular object being dragged is one of a set of application-supplied rendering mechanisms and formats.
DrgVerifyRMF	Verifies if an application-specified rendering mechanism and format is valid for a particular object being dragged.
DrgVerifyTrueType	Verifies if an application-specified type is the true type of a particular object being dragged.
DrgVerifyType	Verifies if an application-specified type is valid for a particular object being dragged.
DrgVerifyTypeSet	Returns the intersection between the contents of the string represented by the type string handle and an application-supplied type string.

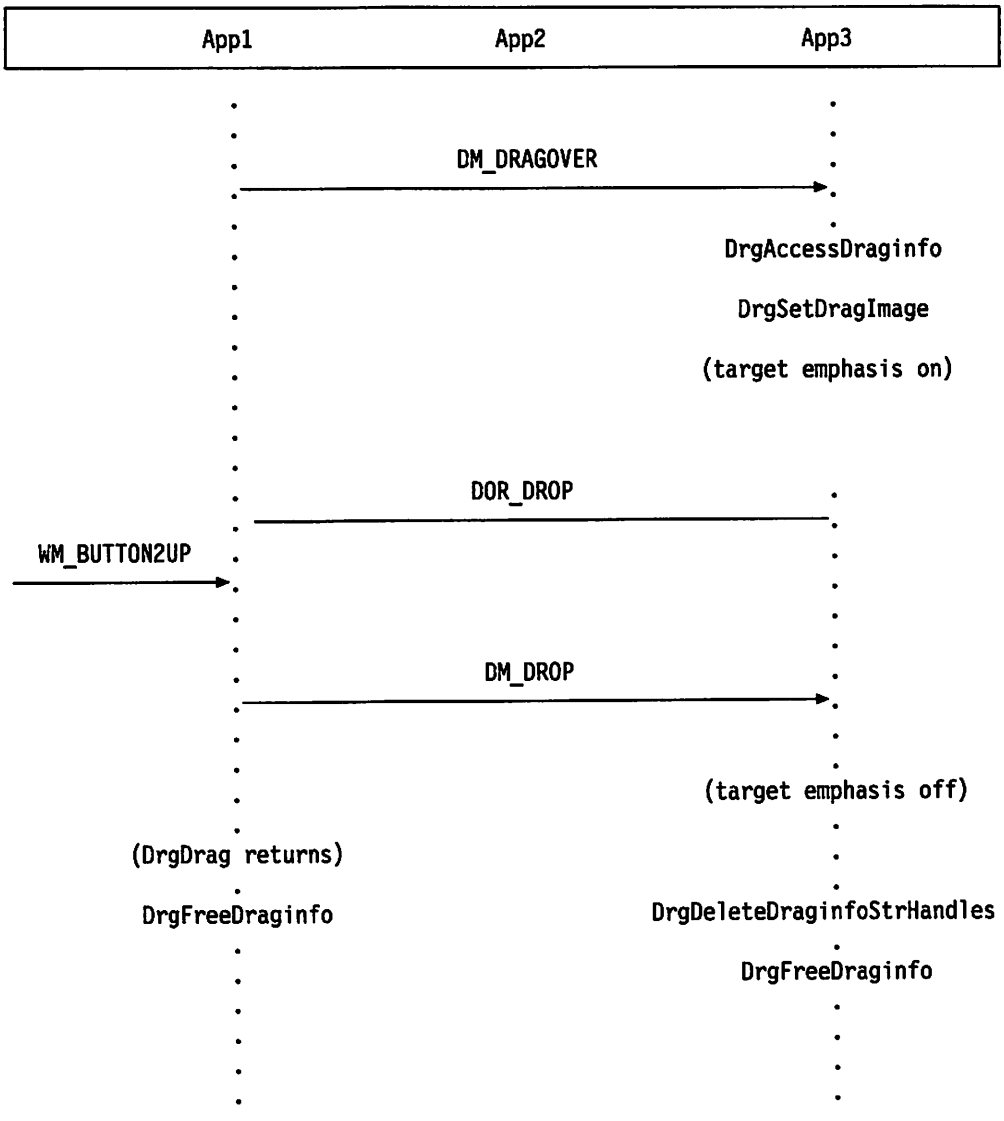
Example - Two Object Drag

The following diagram represents the sequence of API functions and message flows for a typical direct-manipulation operation. The flow shows a two-object drag from App1 to App3, dragging over App2 before reaching App3.

For this example, assume that App1 is implementing the Button 2 drag model.



Two Object Drag (continued)



Application Interaction after a Drop

This portion of the document discusses aspects of a direct-manipulation operation that need to be considered after a drop has occurred. See "Example - Conversation after the Drop" on page 33 for an example.

Conversation Initiation

Direct manipulation offers various ways for both a source and target application to exchange data. To accomplish the exchange, a separate conversation must be established to transfer each data object from the source to the target. It is the responsibility of the target to inform the source about the rendering mechanism it wishes to use and the format the data is to be exchanged in. The target may establish the conversations to run in parallel, or it may initiate the conversations in a serial fashion. The following sections explain how each conversation is established.

Considerations when Establishing a Conversation

A source application may be able to exchange data with a target through several mechanisms. These mechanisms could include:

- Dynamic Data Exchange
- OS/2 File
- Print

Additionally, the source application may be able to render the data into various formats. For example, a spreadsheet application may be able to render its contents in a spreadsheet format or in a text format. Finally, the ability of the source application to render the data in some format, might itself depend on the exchange mechanism used. The rendering mechanisms and formats that a source application can support, for each object that was dropped, are provided to the target through the **hstrRMF** field in the DRAGITEM structure.

The first ordered pair in the set of rendering mechanisms and formats that the source application supports is the object's native rendering mechanism and format. This is the mechanism that most-naturally conveys the data; either where it is now, or where it can most easily be put. The format is the format of the data that conveys all information about the data. For example, a spreadsheet cell has a location in a row and column of a spreadsheet. Rendering the spreadsheet cell in a simple text format would cause this information to be lost, so a more appropriate format should be chosen for its native rendering format.

The target application also may be able to exchange data with the source through several different combinations of mechanism and format. It is the responsibility of the target to obtain the data from the source in the format that they both support and that provides the highest level of information about the data.

While making this determination, the target must consider the exchange capabilities offered by the mechanism. For example, an OS/2 File exchange mechanism can provide only a snapshot of the data at the time the direct-manipulation operation occurred. An exchange using DDE, on the other hand, offers the target an opportunity to remain informed about changes to the data.

Determining if Data Can be Exchanged

During the drag portion of a drag-and-drop operation, the target must determine if it can exchange or receive data from the source for each object involved in the operation. The following minimum requirements must be met to exchange data for an object.

- The source and target must share knowledge of at least one common type for the object. The target can make this determination by using the DrgVerifyTypeSet or DrgVerifyType function.
- The source and target must share at least one common rendering mechanism and format for that type object. The target can make this determination by using the DrgVerifyRMF function.

When these conditions are met, a target can allow the object to be dropped.

Determining How To Exchange the Data

The target makes its determination as to the best rendering mechanism and format to use in the following manner:

1. Uses the native rendering mechanism and format whenever possible.

   This rendering conveys *ALL* information about the data. A target can determine if it supports the native rendering mechanism and format through the use of the following functions:

   - DrgVerifyNativeRMF
   - DrgQueryNativeRMFLen
   - DrgQueryNativeRMF

   Regardless of whether the native rendering mechanism and format supported by the source can be used, the target can elect to exchange the data in a rendering mechanism and format that conveys less information about the object.

2. Uses the next best rendering mechanism and format.

   This is especially true for a Copy operation, because the user will not lose data about the object, as will occur when the object is moved.

   The target can determine the next best rendering mechanism and format to use through repeated calls to the DrgVerifyRMF function. The calls are made starting with the most desirable rendering mechanism and format pair, and progressing to the least desirable pair. Once a pair that the source supports has been found, the target can exchange the data.

Performance Considerations

When context information about an object will be lost because of using a less desirable rendering mechanism and format, the target may elect to pick a common mechanism and format that will achieve the best performance. This is done in the same way that the next best rendering mechanism and format is selected, proceeding from the best-performing rendering to the worst.

Using Direct-Manipulation Data Transfer in an Application

Some standard rendering mechanisms are defined by this system, but this system allows the set of rendering mechanisms to be expanded, allowing for:

- Additional standard rendering mechanisms to be defined in the future
- Application definition of private, or nonstandard rendering mechanisms.

An application may elect to support some, all, or none of the standard rendering mechanisms defined by the system. Applications that do not support any of the standard rendering mechanisms are not precluded from using direct-manipulation. However, support of the standard rendering mechani9ms and formats increases the chances of a successful data transfer between applications.

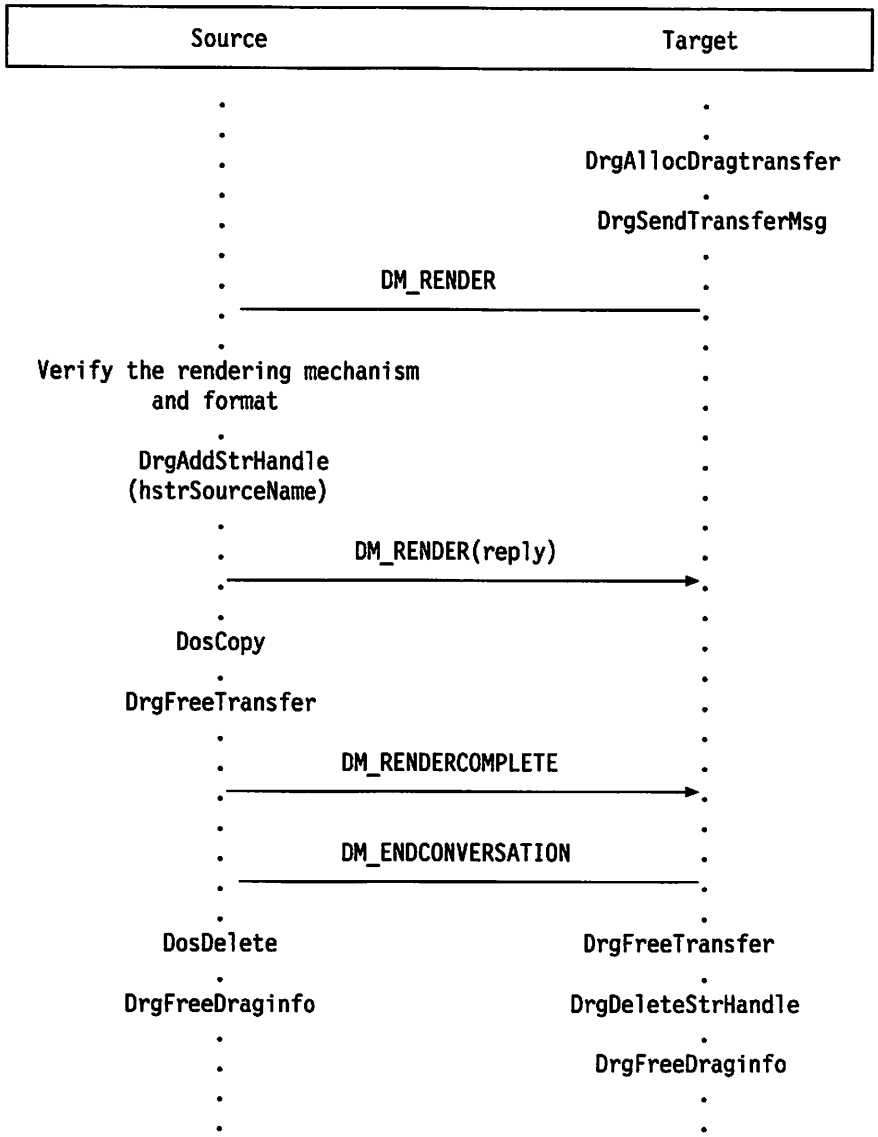
An application that supports a particular rendering mechanism, whether or not it is a rendering mechanism defined by the system, must follow a specific set of guidelines defined by that rendering mechanism, including conversation-initiation procedures and naming conventions. The guidelines for the current system-defined rendering mechanisms are described in the following sections.

Regardless of the rendering mechanism used, it may be necessary to prepare the source for the rendering of the object. Such an action would be necessary when a window needs to be created by the source in order to handle the conversation. This is done by sending a DM_RENDERPREPARE message to the **hwndSource** window in the DRAGINFO structure. This message need only be sent when the DC_PREPARE flag is on in the **fsControl** field of the DRAGITEM structure. When the source receives this message, it will perform any necessary preparation for the rendering and will fill in the **hwndItem** field in the DRAGITEM structure, allowing the target to establish conversation with that window.

Example - Conversation after the Drop

The following diagram represents the sequence of message flows for a typical direct-manipulation data-transfer operation. The flow will describe a single-object move from Source to Target. The user dropped on white space in the target container.

For this example, assume that the rendering mechanism selected is DRM_OS2FILE and that the source does not initially provide the target with the source item's file name. Also assume that the source and target items exist on different drives.



Standard Rendering Mechanisms

OS/2 File

This rendering mechanism might be used by various containers, including file folders and trash cans. These containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They also may allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish some operation.

Mechanism Name: The string for this rendering mechanism is DRM_OS2FILE.

Messages: The following messages are used by the DRM_OS2FILE:

- DM_RENDER

  This message is sent by a target to a source to request a rendering for an object. When this message is received, the source determines if it understands the rendering mechanism and format selected by the target for the object. It also confirms that it allows the operation selected by the user, for that object. The source must respond to this message before proceeding with the rendering operation.

- DM_RENDERCOMPLETE

  This message is posted by a source to a target to notify the target that the rendering operation has been completed by the source, either successfully or unsuccessfully. The source can elect to allow the target to retry a successful or unsuccessful operation. In this case, it should return to its state at the time of the drop for that object and indicate in the message that a retry is allowed.

  Support for this message by a source is optional. If this message is not supported then:

  - The source must convey all necessary information to the target in order to allow it to handle the rendering operation.
  - It must always indicate that native rendering is allowed when replying to a DM_RENDER message.

- DM_ENDCONVERSATION

  This message is sent by a target to a source to notify the source that the rendering operation is complete and that the conversation is terminated. When this message is received, the entire drop operation for the object is complete. The source can now release any resources it had allocated to the drop and rendering operations. When the reply is received, the target may release the resources it had allocated to the operation.

Native Mechanism Actions

Native Rendering by the Target: If the target understands the native rendering mechanism and format of the object, it may be possible to render the object without any involvement on the part of the source, provided the source has given the target sufficient information to do so. In order for the rendering to be performed by the target, the source must fill in, at a minimum, the **hstrContainerName** and **hstrSourceName** fields. This **hstrContainerName** field represents the subdirectory that the file indicated by **hstrSourceName** is in. In order for the target to do the rendering on its own, the true type of the object must be DTYP_OS2FILE. When

these conditions are met, the target may proceed with the operation. When the operation is complete, the target must send a DM_ENDCONVERSATION message to the window indicated by **hwndItem** in the DRAGITEM structure.

Preventing a Target from Rendering an Item: A source can prevent a target from doing the rendering operation on its own by not providing the source name for the object. This may be a necessary action for sources that implement some type of security, or may not allow particular operations to be performed for an object move. When a source takes this course, it must fill in the **hstrSourceName** in the DRAGITEM structure before replying to a DM_RENDER message. The target will delete the **hstrSourceName** string handle prior to freeing the DRAGINFO structure, just as it would if the information had been passed to it at the time of the drop.

Requesting the Source to Render the Item: Whenever the conditions for a target to do the rendering operation without source participation are not met, the target must request the source to carry out the rendering by posting a DM_RENDER message to the source. Of course, the target is free to do this even if it is able to carry out the rendering mechanism on its own.

Allocating and Freeing a DRAGTRANSFER Structure: The data in a drag transfer message is carried in a DRAGTRANSFER structure. DRAGTRANSFER structures are allocated when the target calls DrgAllocDragtransfer.

When the conversation or conversations have been completed, both the source and the target must call DrgFreeDragtransfer to free the shared memory. The target should do it immediately after sending a DM_ENDCONVERSATION message. The source should do it immediately after sending a DM_RENDERCOMPLETE message.

Operation Specifics: Regardless of the operation being performed, the target must fill in the **hstrRenderToName** field in the DRAGTRANSFER structure before sending a DM_RENDER message. This is the fully qualified drive, path, and file name of the file that will contain the data when the rendering operation is complete. When the source has completed the operation, it must post a DM_RENDERCOMPLETE message to the target. The target then must complete the direct-manipulation operation for that object by posting a DM_ENDCONVERSATION message to the source. Once the conversations for all of the objects involved in the drop are complete, the target can delete the string handles and free the DRAGINFO structure.

Non-Native Mechanism Actions: The target may select the DRM_OS2FILE rendering mechanism when it is not the native rendering mechanism for an object, as long as the source supports it. In this case, the target must always request that the source carry out the rendering operation as described above. The source should render the data in the requested format to the file specified by the **hstrRenderToName** field. If the requested operation is a Move, the source should take whatever action is necessary to remove its knowledge of the object as long as no information regarding the object was lost in the transfer.

Naming Conventions: The naming conventions for this rendering mechanism follow:

- hstrContainerName

  Contains the fully qualified drive and path name for the source file.

  Examples are:

```
C:\
C:\MYSUBDIR\
A:\SUBDIR1\SUBDIR2\
\\NETWORK\SHARED\SUBDIRA\SUBDIRB\
```

- hstrSourceName

  Contains the name of the source file or subdirectory. For example:

  ```
  MYSOURCE.C
  MYSOURCE.H
  MYSOURCE IS A LONG FILE NAME
  SUBDIR3
  ```

- hstrRenderToName

  Contains the fully qualified file or subdirectory name that is to be used at the target. For example:

  ```
  C:\MYSUBDIR\MYSOURCE.C
  \\NETWORK\SHARED\SUBDIRA\SUBDIRB\MYSOURCE.H
  C:\SUBDIR1\SUBDIR2\SUBDIR3
  ```

Types: Any type that is allowed as a *.TYPE* extended attribute is allowed in the **hstrType** field of the DRAGITEM structure. The typ8 for a file may be obtained using the DosQFileInfo function, and set by using the DosSetFileInfo function.

Print

A common object that might be provided by a container is a printer. This object would allow objects to be dragged and dropped on it to accomplish a print operation.

Mechanism Name: The string for this rendering mechanism is DRM_PRINT.

Messages: To support this rendering mechanism, a source must be able to receive and process a DM_PRINT message. The target will post this message to the source. When the message is received, the source prints the current view of the object identified in the message to the printer queue, which is also identified in the message.

Native Mechanism Actions: There are no native mechanism actions for this rendering mechanism, because the act of printing an object is considered a transform from the native rendering mechanism to the print mechanism.

Naming Conventions: None.

DDE

This rendering mechanism would be used by various containers and applications. The containers allow objects to be dragged and dropped on white space in the container to accomplish a Move or Copy operation. They also may allow objects in the same or another container to be dragged and dropped on objects within the container to accomplish some operation.

Mechanism Name: The string for this rendering mechanism is DRM_DDE.

Messages: To support this rendering mechanism a source must be able to receive and process the following messages:

- WM_DDE_REQUEST

This message is posted by the target to the window indicated by the **hwndItem** field in the DRAGITEM structure in order to request information regarding the object. Note that WM_DDE_INITIATE is not required because the target already has the handle of the window with which it wishes to converse. This message is sent for all Move and Copy operations.

- WM_DDE_ADVISE

  This message is posted by the target to the window indicated by the **hwndItem** field in the DRAGITEM structure in order to maintain a hot link to the object.

- WM_DDE_UNADVISE

  This message is posted by the target to the window indicated by the **hwndItem** field in the DRAGITEM structure in order to terminate a hot link to the object.

- WM_DDE_TERMINATE

  This message is posted by the target to the window indicated by the **hwndItem** field in the DRAGITEM structure in order to terminate a conversation.

In order to support this rendering mechanism, a target must be able to receive and process the following messages:

- WM_DDE_DATA

  This message is posted to the target by the source in order to deliver the requested information regarding the object.

- WM_DDE_ACK

  This message is posted to the target by the source to acknowledge a WM_DDE_ADVISE or WM_DDE_UNADVISE message.

- WM_DDE_TERMINATE

  This message is posted to the target by the source in order to end a conversation.

Native Mechanism Actions: Prior to establishing a DDE conversation, the target should determine the source-supported formats in which it wishes to have the object rendered. It should register this format in the system atom table, and use the resulting atom in the **usFormat** field of the DDESTRUCT used in the conversation.

The target should establish the DDE conversation by posting a WM_DDE_REQUEST message to the window indicated by the **hwndItem** field in the DRAGITEM structure. The target acts as the client, and the source acts as the server in the conversation.

Operation Specifics: The following actions should be taken by the source, depending on the operation being performed:

- Copy: Simply send the data to the target.
- Move: Remove knowledge of the object after receiving confirmation that the target has successfully completed its portion of the rendering operation.

Non-Native Mechanism Actions: The target and source proceed in the same way, regardless of whether DDE was the native rendering mechanism or an alternate rendering mechanism.

Naming Conventions: The naming conventions for the DRM_DDE rendering mechanism follow:

- hstrSourceName

  Contains the object name to be used in the DDE conversation.

- hstrRMF

  The format portion of the list of ordered pairs in the format *<DRM_DDE,format>* identifies the formats supported by the source for the object. The non-standard DDE formats that these formats map to, must be registered in the system atom table by both the source and the target.

Types: Any type that is allowed as a *.TYPE* extended attribute is allowed in the **hstrType** field of the DRAGITEM structure.

Application Extensions to the Direct-Manipulation Data-Transfer Protocol

An application may choose to define a new rendering mechanism. However, if an application intends to provide renderings from this extended rendering mechanism to existing rendering mechanisms, it should publish enough information so that other application developers can use the new mechanism. An application must address several distinct areas of definition. These areas are described below, in general, and also are addressed under the definition for the system mechanisms.

Rendering Mechanism Name

The string name of the rendering mechanism should be defined by the application. This string name will be specified in the mechanism/format pair of the DRAGITEM structure.

Native Mechanism Actions

When both a source and target application store the data in the same native mechanism, a transform is not required. Instead, the native Move and Copy actions for that mechanism could be performed by the target. An application must completely define the proper procedure for performing that action. In the case of files, the native move action is defined as a DosMove or DosCopy/DosDelete. The native copy action is DosCopy. An application need not support all of the basic actions; it may choose to define additional native mechanism actions, indicated by the DO_UNKNOWN action in the DRAGINFO structure.

Naming Conventions

An application that is defining a new mechanism must completely specify the naming conventions for objects rendered in that mechanism. This information typically includes both the name of the data and preceding information describing the exact location of the data. Any special rules concerning uppercase and lowercase or character sets to be used in naming must also be specified. The semantics for using these mechanism names, as well as an algorithm for deriving location information, also must be defined.

An application that is defining a new rendering mechanism must completely define the set of messages that a target and source application must support and specify the appropriate action to be taken for each message. The message IDs (above WM_USER) for the messages must be published.

Performance Considerations

If an application provides or defines transforms from the newly defined mechanism to existing mechanisms, performance information about the transform between mechanisms should be provided. This will aid the application developer in choosing the appropriate transform when it encounters an application that transforms from an unknown native mechanism to several different known mechanisms.

Chapter 30. Alphanumeric Video Output

Summary of Changes

New	Updated	Section Title
	√	Setting the Cell Size
	√	Setting the Display Mode For Text-Windowed and Full-Screen Applications
	√	Display Mode Attributes Supported by Adapters (table)
	√	Loading Fonts

Setting the Cell Size

On page 30-2, under "Setting the Cell Size", replace the table showing cell sizes supported by adapters, with the following table:

Adapter	Cell Sizes
EGA	8 x 8, 8 x 10, 8 x 12, 8 x 14 (default), 8 x 16, 8 x 18
VGA	8 x 8, 8 x 10, 8 x 12, 8 x 14 (default), 8 x 16, 8 x 18
8514/A	6 x 14, 7 x 12, 7 x 15, 7 x 25, 8 x 8, 8 x 12, 8 x 14, 8 x 17 (default), 12 x 16, 12 x 20, 12 x 22, 12 x 30
XGA	6 x 10, 6 x 14, 7 x 15, 7 x 25, 8 x 8, 8 x 10, 8 x 12, 8 x 14, 8 x 16, 8 x 18, 10 x 18, 12 x 16, 12 x 20, 12 x 22, 12 x 30 (Where 8 x 14 is the default for a 640 x 480 resolution monitor, and 12 x 22 is the default for a 1024 x 768 resolution monitor.)

Setting the Display Mode For Text-Windowed and Full-Screen Applications

On page 30-7, add a line to the table showing which row values are supported by various adapters for text-windowed applications, for the XGA adapter:

Adapter	Rows
XGA	25, 30, 43, 50 or 60

On page 30-7, replace the last paragraph on the page with the following:

To set the mode, initialize the data structure and issue **VioSetMode**. VioSetMode initializes the current settings for cursor position and type; however, it does not clear the screen (except in DOS mode only). To clear the screen, you must issue one of the VioScrollXX calls. See "Scrolling the Screen Image." For information about resetting the cursor position and type, see "Manipulating the Cursor" on page 30-6.

Display Mode Attributes Support by Adapters (table)

On page 30-8, make the following changes to Table 30-1:

In the last 2 lines of the table add XGA/HColor to the list of Valid Adapter/Display Combinations.

In the previous 2 lines of the table add XGA/Color to the list of Valid Adapter/Display Combinations.

In the list of Display combinations appended to the table, redefine Color to be 8512/13 PS/2 Color Display, 8514 Color Display; and redefine HColor to be 8514 Color Display and 8515 Color Display.

Loading Fonts

On page 30-9, add a line to the table showing which ROM font sizes are supported by various adapters, for the XGA adapter:

Adapter	**Font Sizes Supported**
XGA	8 x 8, 8 x 14, 9 x 14, 8 x 16, 9 x 16

Begin 1990 update for Memory Management

Chapter 32. Memory Management

Summary of Changes

New	Updated	Section Title
	√	Allocating a Locally Shared Segment
	√	Creating and Accessing Discardable Segments

Allocating a Locally Shared Segment

On page 32-3, replace the paragraph preceding the **Note** with the following:

A process may issue DosAllocSeg or DosAllocHuge to allocate shareable segments of memory. The segment may be shareable through DosGiveSeg or DosGetSeg.

After a process allocates a segment or multiple segments, and designates the memory as shareable through DosGiveSeg, the process can then call DosGiveSeg to enable a child process to share the segment. The process passes the segment's selector and the process ID of the intended sharer to the system.

On page 32-3, replace the paragraph after the **Note** with the following:

After a process allocates a segment or multiple segments and designates the memory as shareable through DosGetSeg, the process that allocated the memory (the owning process) must pass the selector of the segment to the sharing process using some means of interprocess communication.

Creating and Accessing Discardable Segments

On page 32-4, after the third paragraph, insert the following:

Issue DosUnlockSeg when you want a swappable segment to be the next segment swapped out of memory. This is particularly important when creating program data such as control blocks or instance data that are not immediately required, but will be required in the future. You can mark those segments with DosUnlockSeg and have them swapped out first when the physical memory in your machine is over-committed. This produces efficient use of memory and increases the performance of a system operating in a memory-constrained environment.

Chapter 33. File and Disk Management

Summary of Changes

New	Updated	Section Title
	√	Controlling Access to EAs
	√	Defining EA data structures

Controlling Access to EAs

On page 33-23, replace the second paragraph with the following:

In addition, operations on extended attributes (EAs) are not atomic: any one query or set operation may not complete before any other query or set operation can be performed on the same object. For example, if an error occurs before an entire list of EAs has been set, all, some, or none of them may actually remain set on the file object. This means that EAs may not remain in a consistent state unless the order in which the operations are performed can be guaranteed.

Defining EA data structures

On page 33-25, replace **Figure 33-8. EA data structures** with the following:

```
struct FEA
       unsigned char    flags;          /* byte of flags                */
       unsigned char    cbName;         /* length of EA name            */
       unsigned short   cbValue;        /* length of EA value           */
       unsigned char    szName[];       /* ASCII EA name                */
       unsigned char    aValue[];       /* free-format EA value         */
       ;

struct FEAList
       unsigned long    cbList;         /* length of list               */
       struct FEA list[];               /* set of FEAs                  */
       ;

struct GEA
       unsigned char    cbName;         /* length of EA name            */
       unsigned char    szName[];       /* ASCII EA name                */
       ;

struct GEAList
       unsigned long    cbList;         /* length of list               */
       struct GEA list[];               /* set of GEAs                  */
       ;

struct EAOP
       struct GEAList far * fpGEAList;  /* set of GEAs                  */
       struct FEAList far * fpFEAList;  /* set of FEAs                  */
       unsigned long  offError;         /* offset of FEA err            */
       ;
```

Chapter 36. Printing

Summary of Changes

New	Updated	Section Title
√		Spooler

Spooler

The OS/2 Version 1.3 spooler offers improvements in system utilization and performance. It implements the DosPrint application programming interface for ease of migration of DOS applications, and for improved access to spooler facilities.

The OS/2 Version 1.3 spooler extends the function of the DosPrint spooler calls supplied by Microsoft* with their LAN Manager Version 1.0 Applications Program Interface (API), and used in other applications, to control the print spooling of servers on a local area network (LAN). The OS/2 Version 1.3 spooler APIs allow the application to control the Presentation Manager* print spooling system both locally and on the network.

The DosPrint API types and constants are defined in PMSPL.H. For a description of the data types, structures, and calls that are referred to, but not described, in this section, see the *Presentation Manager Programming Reference* in the *IBM OS/2 Version 1.2 Programming Tools and Information* library. For descriptions of these data types, structures, and calls in the C language, see the *Presentation Manager C/2 Bindings Reference* in the same library.

Spooler Calls

The functional group to which a particular call belongs is identified in the name of the call. The spooler calls comprise three functional groups, indicated by the first part of the call name:

Functional Group:	Call Name Starts:
Print destination calls	DosPrintDest
Print job calls	DosPrintJob
Print queue calls	DosPrintQueue

The action performed by a call is indicated by the final part of the call name. The main generic calls are :

Add An Add call adds a resource to a particular set of items.

Del A Del call removes a resource from a set of items.

Enum An Enum call lists information about system resources. The parameters usually include Buf, Length, Returned, and Total.

If Buf cannot store the fixed-length returned data, the Enum call returns NERR_BufTooSmall. In this case, all data in Buf is invalid, but Total contains the number of resources available.

If Buf can store all the fixed-length returned data but not all the available variable-length data, the Enum call returns ERROR_MORE_DATA. In this

case, the fixed-length data in Buf is valid, with pointers to any incomplete variable-length data set to NULL.

If the value of Length is 0, the Enum call returns only a valid Total parameter.

GetInfo A GetInfo call retrieves specific information about a resource. The parameters usually include Buf, Length, and pcbNeeded.

If Buf cannot store the fixed-length returned data, the GetInfo call returns NERR_BufTooSmall. In this case, all data in Buf is invalid, but Needed contains the number of bytes necessary to store the information.

If Buf can store the fixed-length returned data but not all the available variable-length data, the GetInfo call returns ERROR_MORE_DATA. In this case, the fixed-length data in Buf is valid, with pointers to any incomplete variable-length data set to NULL.

If the value of Length is 0, the GetInfo call returns only a valid Needed parameter.

SetInfo A SetInfo call sets the parameters of a resource.

The parameter ParmNum specifies whether only one specific component of the data structure is passed in Buf and is to be changed, or the entire data structure is passed in Buf (in this case ParmNum is 0).

A GetInfo call can be used first, and the required changes made to the resulting contents of Buf. Care must be taken to ensure that changed pointer components no longer point into the buffer, or they will be assumed to be unchanged.

The following specialized actions are also indicated in the name of the relevant call:

Continue	Continues a paused print job or queue
Control	Pauses, continues, or restarts a print destination, or deletes the current job
GetId	Retrieves a job identifier
Pause	Pauses a print job or queue
Purge	Deletes all entries in a queue.

Print Destination Calls

The spooler printer defined in the PM_SPOOLER_PRINTER section of OS2.INI is used as the Presentation Manager spooler equivalent of a print destination. An entry in PM_SPOOLER_PRINTER maps one-to-one onto a logical address, called a PM_SPOOLER_PORT.

The following print destination calls are provided for use on a local workstation or, if the LAN program is installed, on a remote server:

DosPrintDestAdd	This new call establishes a print destination.
DosPrintDestControl	This call cancels, pauses, continues, or restarts a print destination. It uses the logical address to refer to the destination printer.
DosPrintDestDel	This new call deletes a print destination.
DosPrintDestEnum	This call obtains information about all print destinations on a computer.
DosPrintDestGetInfo	This call retrieves information about a particular print destination.

DosPrintDestSetInfo This new call modifies the configuration of a print destination.

The *PRDINFO3* data structure is used for these calls at level 3. Its use is indicated by a value of 3 for the **level** parameter.

Print Job Calls

These calls manipulate individual print jobs in a print queue.

Print jobs are identified by an integer job identifier (**Job**). This identifier is unique on a particular computer, not only within a queue. A combination of computer name and job identifier is sufficient to uniquely identify any job on a network. A job is assigned a job identifier by the spooler when the job is queued. The following print destination calls are provided for use on a local workstation or, if the LAN program is installed, on a remote server:

DosPrintJobContinue This call continues a paused print job. A paused print job remains in the queue but is not selected to print. If it reaches the front of the queue, the spooler bypasses it and prints jobs from behind it until the job is reactivated by a DosPrintJobContinue call.

DosPrintJobDel This call removes a print job from a print queue, canceling the printing.

DosPrintJobEnum This call obtains information about all the jobs in a print queue.

DosPrintJobGetId This call retrieves the job identifier of a remote print job.

DosPrintJobGetInfo This call retrieves information about a particular print job.

DosPrintJobPause This call pauses a job in a print queue. A paused print job remains in the queue but is not selected to print. If it reaches the front of the queue, the spooler bypasses it and prints jobs from behind it until the job is reactivated by a DosPrintJobContinue call.

DosPrintJobSetInfo This call modifies the instructions for a print job. It can be used to give a particular print job priority over other print jobs by changing the position of the job in the print queue, or its priority. An application that does not have administrator privilege can only move jobs backwards in a remote print queue, or set a priority lower than the queue priority.

The *PRJINFO3* data structure provides full job details for those applications that need them. Its use is indicated by a value of 3 for the **level** parameter. The *PRJINFO2* structure provides a subset of the information supplied by *PRJINFO3*; it minimizes the storage required for job-information retrieval, and is sufficient for most uses. Its use is indicated by a value of 2 for the **level** parameter.

The *PRIDINFO* data structure provides information returned by a DosPrintJobGetId call about a remote print job.

Print Queue Calls

The following print queue calls are provided for use on a local workstation, or on a remote server if the LAN program is installed :

DosPrintQAdd This call creates a new print queue.

DosPrintQContinue This call continues a paused print queue.

DosPrintQDel This call deletes a print queue.

DosPrintQEnum This call retrieves information about all print queues on a

DosPrintQGetInfo This call retrieves information about a particular print queue, and, optionally, about the jobs in it.

DosPrintQPause This call pauses a print queue.

DosPrintQPurge This call removes all jobs from a print queue.

DosPrintQSetInfo This call modifies the configuration of a print queue.

The *PRQINFO3* data structure is used for these calls at level 3. Its use is indicated by a value of 3 for the **level** parameter.

Existing Applications and Migration

The design allows many existing applications to continue to work without modification, but they must be relinked with OS2.LIB because the application programming interfaces, which were in NETSPOOL.DLL, are now in PMSPL.DLL. Applications that need to be recompiled or relinked for other reasons are supported by supplied header files. No DosPrint changes are necessary unless new functionality is required.

Previous information levels for all API calls are still supported. In addition, the new information levels will work when directed to servers of previous LAN Manager versions. New fields are either filled with benign values, or given a value that indicates that they are not available.

It is not possible to provide full compatibility with existing applications that exploit all the features of the DosPrint calls in the Microsoft OS/2 LAN Manager API.

Existing Print Destination Calls

Existing DosPrintDest applications use the *PRINTDEST* data structure. This is replaced by the *PRDINFO* data structure, which maps element for element onto the *PRINTDEST* structure:

```
typedef struct _PRDINFO {
 szName;
 szUserName;
 uJobid;
 fsStatus;
 pszStatus;
 time;
 } PRDINFO;
```

```
struct PRINTDEST {
 prdest_name;
 prdest_username;
 prdest_jobid;
 prdest_status;
 prdest_status_string;
 prdest_time;
}
```

New applications should not use the *PRDINFO* data structure, but should take full advantage of the additional information provided by the *PRDINFO3* structure.

Compatibility

- DosPrintDestEnum, DosPrintDestGetInfo
  These calls accept or return information at levels 0 and 1 for compatibility with existing applications.

Existing Print Job Calls

Existing applications use the *PRINTJOB* data structure. This is replaced by the *PRJINFO* data structure, which maps element for element onto the *PRINTJOB* structure:

```
typedef struct _PRJINFO {
 uJobid;
 szUserName;
 pad_1
 szNotifyName;
 szDataType;
 pszParms;
 uPosition;
 fsStatus;
 pszStatus;
 ulSubmitted;
 ulSize;
 pszComment;
 } PRJINFO;
```

```
struct PRINTJOB {
 prjob_id;
 prjob_username;
 prjob_pad_1;
 prjob_notifyname;
 prjob_datatype;
 prjob_parms;
 prjob_position;
 prjob_status;
 prjob_status_string;
 prjob_submitted;
 prjob_size;
 prjob_comment;
}
```

New applications should not use the *PRJINFO* data structure, but should take full advantage of the additional information provided by the *PRDINJO2* and *PRDINJO2* structures.

Compatibility

- DosPrintJobEnum
  Calls at levels 0 and 1 are compatible with existing applications, with these exceptions:
  - The job data type may be truncated to fit the *PRJINFO* structure.
  - The number of destination status bits is 8. New applications have access to 11 bits to enable them to determine the cause of a print job failure (for example, waiting for a form change).
- DosPrintJobGetInfo
  Calls at levels 0 and 1 are compatible with existing applications, with these exceptions:
  - The job data type may be truncated to fit the *PRJINFO* structure.
  - The number of destination status bits is 8. New applications have access to 11 bits to enable them to determine the cause of a print job failure (for example, waiting for a form change).

  A job with a status of PRJ_QS_SPOOLING_ cannot be listed, because the OS/2 Presentation Manager spooler does not recognize this concept.

Existing Print Queue Calls

Existing applications use the *PRINTQ* data structure. This is replaced by the *PRQINFO* data structure, which maps element for element onto the *PRINTQ* structure:

```
typedef struct _PRQINFO {          struct PRINTQ {
 szName;                            prq_name;
 pad_1                              prq_pad;
 uPriority;                         prq_priority;
 uStartTime;                        prq_starttime;
 uUntilTime;                        prq_untiltime;
 pszSepFile;                        prq_separator;
 pszPrProc;                         prq_processor;
 pszDestinations;                   prq_destinations;
 pszParms;                          prq_parms;
 pszComment;                        prq_comment;
 fsStatus;                          prq_status;
 cJobs;                             prq_jobcount;
 } PRQINFO;                        }
```

New applications should not use the *PRQINFO* data structure, but should take full advantage of the additional information provided by the *PRQINJO3* structure.

The **prproc** field used to contain the path to a .EXE file. Queue processors are now dynamic link libraries, and the **prproc** field must now contain the name of the required queue processor.

If a value is found in this field, it is compared with the names of the defined queue processors (for example, PM_SPOOLER_QP). If it is not found, ERROR_INVALID_PARAMETER is returned. If a NULL pointer is passed in the **prproc** field when a queue is created, or a pointer to an empty string in DosPrintQSetInfo, the default queue processor is selected.

Compatibility

- DosPrintQAdd
  Existing applications that create queues can use a uLevel value of 1 and pass a *PRQINFO* structure into DosPrintQAdd. These restrictions apply:
  - the **qproc** field contains the name of the queue processor, not the path to a .EXE file.
  - Existing applications must specify a list of ports in the **destinations** field. The port list might be NULL, in which case the queue is not connected. If a list of ports is defined, and these ports are named by spooler printers, the queue is created to print on those printers.
  - If any specified port is not referenced in PM_SPOOLER_PRINTERS, the port is not created, and the queue is not connected.
  - Devices are separated by a space.
- DosPrintQEnum
  Calls at levels 0, 1, and 2 are for compatibility with existing applications, but should not be used by new applications. Some elements of the *PRQINFO* structure now have different values in level 1 and 2 calls:
  - The **prproc** field is returned containing the name of the Presentation Manager queue processor (usually PMPRINT), and not the path to a .EXE file as before.

- The **destinations** field lists the PM_SPOOLER_PORTS that the queue can print on. This can now be NULL, in which case the Destinations pointer in the returned data structure is NULL.

Levels 0, 1, and 2 do not return queue names that are longer than QNLEN. The API filters these queues, and **Total** is set to the number of queues whose names do not exceed QNLEN.

- DosPrintQGetInfo
  Calls at levels 0, 1, and 2 are for compatibility with existing applications. They do not provide all the available information, and should not be used by new applications. Some elements of the *PRQINFO* structure now have different values in level 1 and 2 calls:
  - The **prproc** field is returned containing the name of the Presentation Manager queue processor (usually PMPRINT), and not the path to a .EXE file as before.
  - The **destinations** field lists the PM_SPOOLER_PORTS that the queue can print on. This can now be NULL, in which case the Destinations pointer in the returned data structure is NULL.

  Levels 0, 1, and 2 do not return queue names that are longer than QNLEN. The API filters these queues, and **Total** is set to the number of queues whose names do not exceed QNLEN.

- DosPrintQSetInfo
  The PRQINFO structure and level 1 call is for compatibility with existing applications, and should not be used by new applications. At level 1 some fields have different meanings in the Presentation Manager spooler:
  - The **prproc** field now contains the name of the Presentation Manager queue processor (usually PMPRINT), and not the path to a .EXE file as before. An existing application that specifies a path will get the error ERROR_INVALID_PARAMETER.
  - Existing applications must specify a list of ports in the **destinations** field. The port list might be NULL, in which case the queue is not connected. If a list of ports is defined, and these ports are named by spooler printers, the queue is created to print on those printers.
  - If any specified port is not referenced in PM_SPOOLER_PRINTERS, the port is not created, and the queue is not connected.
  - Devices are separated by a space.

Down-Level Support

Previous information levels for all API functions are supported. In addition, the new information levels will work when directed to servers using earlier versions of the Microsoft OS/2 LAN Manager API. New fields will be filled with some benign value or given a value that means 'not available'. Specific fields that are simulated or unavailable are:

- DosPrintJobGetInfo, DosPrintJobEnum
  At level2:

uPriority	Always PRJ_NO_PRIORITY (0)
pszDocument	NULL

  At level 3:

pszQProcName	NULL
pszQProcParms	NULL
pszQDriverName	NULL

pDriverData	NULL
pszQPrinterName	NULL.

- DosPrintJobSetInfo
  The following values for ParmNum return ERROR_NOT_SUPPORTED:

  PRJ_PRIORITY_PARMNUM
  PRJ_PROOCPARMS_PARMNUM
  PRJ_DRIVERDATA_PARMNUM

  If the entire structure is passed, unsupported entries are ignored, no error is returned, and the fields are not checked for validity.

- DosPrintQAdd, DosPrintQEnum, DosPrintQGetInfo
  At level 3:

pszPrProc	If no print processor is specified, 'LM10' is returned as the print processor.
pszPrinters	Returns a list of ports separated by commas (assumes that there are printers named like ports)
pszDriverName	NULL
pDriverData	NULL

  At level 4:

uPriority	Always PRJ_NO_PRIORITY (0)
pszPrProc	If no print processor is specified, 'LM10' is returned as the print processor.
pszParms	NULL
pszPrinters	Returns a list of ports separated by commas (assumes that there are printers named like ports)
pszDriverName	NULL
pDriverData	NULL
pDriverData	NULL.

- DosPrintQSetInfo
  The following values for Parmnum return ERROR_NOT_SUPPORTED:

  PRQ_DRIVERNAME_PARMNUM
  PRQ_DRIVERDATA_PARMNUM

  PRQ_PRINTERS_PARMNUM is converted to PRQ_DESTINATIONS_PARMNUM by replacing some commas with blanks. If the entire structure is passed, unsupported entries are ignored, no error is returned, and the fields are not checked for validity.

Functions with no Down-Level Emulation

These functions have not changed and so need no special down-level support:

DosPrintDestControl
DosPrintDestEnum
DosPrintDestGetInfo
DosPrintJobContinue
DosPrintJobDel
DosPrintJobGetId
DosPrintJobPause
DosPrintQContinue
DosPrintQDel
DosPrintQPause
DosPrintQPurge.

These functions are new and therefore have no emulation:

DosPrintDestAdd
DosPrintDestDel
DosPrintDestSetInfo.

Constants

The constants used by the spooler data types and calls are included here :

```
CNLEN                   15   /* Computer name length */
DTLEN                    9   /* Spool file data type */
PDLEN                    8   /* Print destination length */
QNLEN                   12   /* Queue name maximum length, 8 maximum for PM */
UNLEN                   20   /* Maximum user name length */
MAXCOMMENTSZ            48   /* Queue comment length */

PRD_STATUS_MASK      0x0003   /* Bits 0,1 */
PRD_ACTIVE              0
PRD_PAUSED              1
PRD_DEVSTATUS        0x01FC   /* Bits 2-8 */
PRD_DELETE              0
PRD_PAUSE               1
PRD_CONT                2
PRD_RESTART             3


PRD_LOGADDR_PARMNUM     3
PRD_COMMENT_PRMNUM      7
PRD_DRIVERS_PARMNUM     8
PRD_TIMEOUT_PARMNUM    10

PRJ_QSTATUS          0x0003   /* Bits 0,1 */
PRJ_QS_QUEUED           0
PRJ_QS_PAUSED           1
PRJ_QS_SPOOLING         2
PRJ_QS_PRINTING         3
PRJ_DEVSTATUS        0x0FFC   /* Bits 2-11 */
PRJ_COMPLETE         0x0004   /* Bit 2 */
PRJ_INTERV           0x0008   /* Bit 3 */
PRJ_ERROR            0x0010   /* Bit 4 */
PRJ_DESTOFFLINE      0x0020   /* Bit 5 */
PRJ_DESTPAUSED       0x0040   /* Bit 6 */
PRJ_NOTIFY           0x0080   /* Bit 7 */
PRJ_DESTNOPAPER      0x0100   /* Bit 8 */
PRJ_DESTFORMCHG      0x0200   /* Bit 9 */
PRJ_DESTCRTCHG       0x0400   /* Bit 10 */
PRJ_DESTPENCHG       0x0800   /* Bit 11 */
PRJ_DELETED          0x8000   /* Bit 15 */
```

```
PRJ_MAX_PRIORITY         99   /* Lowest priority */
PRJ_MIN_PRIORITY          1   /* Highest priority */
PRJ_NO_PRIORITY           0

PRJ_NOTIFYNAME_PARMNUM    3
PRJ_DATATYPE_PARMNUM      4
PRJ_PARMS_PARMNUM         5
PRJ_POSITION_PARMNUM      6
PRJ_COMMENT_PARMNUM      11
PRJ_DOCUMENT_PARMNUM     12
PRJ_PRIORITY_PARMNUM     14
PRJ_PROCPARMS_PARMNUM    16
PRJ_DRIVERDATA_PARMNUM   18


PRQ_STATUS_MASK      0x0003
PRQ_ACTIVE                0
PRQ_PAUSED                1
PRQ_ERROR                 2
PRQ_PENDING               3

PRQ3_PAUSED          0x0001
PRQ3_PENDING         0x0002

PRQ_MAX_PRIORITY          1   /* Highest priority */
PRQ_DEF_PRIORITY          5
PRQ_MIN_PRIORITY          9   /* Lowest priority */
PRQ_NO_PRIORITY           0


PRQ_PRIORITY_PARMNU       2
PRQ_STARTTIME_PARMNUM     3
PRQ_UNTILTIME_PARMNUM     4
PRQ_SEPARATOR_PARMNUM     5
PRQ_PROCESSOR_PARMNUM     6
PRQ_DESTINATIONS_PARMNUM  7
PRQ_PARMS_PARMNUM         8
PRQ_COMMENT_PARMNUM       9
PRQ_PRINTERS_PARMNUM     12
PRQ_DRIVERNAME_PARMNUM   13
PRQ_DRIVERDATA_PARMNUM   14
```

Spooler Return Codes

The spooler return codes are listed in numerical order with explanations.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_INVALID_HANDLE (6)	An invalid handle is specified.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_NAME (123)	The computer name is invalid.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_DevNotRedirected (2107)	The device is not connected.

NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_DestNotFound (2152)	The printer destination cannot be found.
NERR_QExists (2154)	The printer queue already exists. operations.
NERR_DestInvalid (2159)	This printer destination request contains an invalid control function.
NERR_ProcNoRespond (2160)	The queue processor is not responding.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print destination.
NERR_JobInvalidState (2164)	This operation cannot be performed on the print job in its current state.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_InvalidDevice (2294)	The requested device is invalid.
NERR_InvalidComputer (2351)	The computer name is invalid.

Chapter 39. Supporting National Languages

Summary of Changes

New	Updated	Section Title
	√	Code Page API Summary

Code Page API Summary

On page 39-12, add the following:

DosGetCollate
Get the collating table for the code page that resides in the country information file.

DosGetDBCSEv
Get a double-byte character-set (DBCS) environment vector that resides in the country information file.

DosGetCtryInfo
Get the country dependent information that resides in the country information file.

Appendix G. System Limits

Summary of Changes

New	Updated	Section Title
	√	Limits for Various Operating System Resources

Limits for Various Operating System Resources

On page G-3, replace the table with the following:

Resource	System Limit	Available to Applications
Processes	511	504
Threads	511	483
Threads per process	53	53
Open file handles	64KB	—
Pipes	Part of open file handles	Part of open file handles
Pipe size	64KB	—
Queues	1,448	1,256
Queue priorities	16	—
Queue elements per queue	3,268	—
System semaphores	256	213
Semaphores with outstanding waits	128	126
Maximum semaphores per DosMuxSemWait	16	—
System signals per process	3	—
User signals per process	3	—

Threads:

The minimum available space on the stack for a thread calling OS/2 functions is 4KB. The system allows only one timer per thread.

System Semaphores:

System semaphores have a *use* count and a *reference* count:

- The *use* count is the number of events a system semaphore can represent. A system semaphore is limited to having no more than 255 concurrent uses. A *use* is any one of the following events:
  - Creating a system semaphore.
  - Opening a system semaphore.
  - Specifying the handle of the system semaphore on a call to an OS/2 API, for example, DosTimerAsync.
- The *reference* count is the number of threads waiting for an exclusive system semaphore. A system semaphore is limited to having no more than 255 concurrent references.

Appendix H. Information Presentation Facility Tags

Summary of Changes

New	Updated	Section Title
	√	:hp. (Highlight phrases)

:hp. (Highlight phrases)

On page H-26, replace the syntax table with the following:

Syntax

Tag	Element	Results	End
:hp1.	Highlighted phrase level 1	*Highlight phrase 1 looks like this.*	:ehp1.
:hp2.	Highlighted phrase level 2	**Highlight phrase 2 looks like this.**	:ehp2.
:hp3.	Highlighted phrase level 3	***Highlight phrase 3 looks like this.***	:ehp3.
:hp4.	Highlighted phrase level 4	Color 1 (Blue)	:ehp4.
:hp5.	Highlighted phrase level 5	Highlight phrase 5 looks like this.	:ehp5.
:hp6.	Highlighted phrase level 6	*Highlight phrase 6 looks like this.*	:ehp6.
:hp7.	Highlighted phrase level 7	**Highlight phrase 7 looks like this.**	:ehp7.
:hp8.	Highlighted phrase level 8	Color 2 (Red)	:ehp8.
:hp9.	Highlighted phrase level 9	Color 3 (Pink)	:ehp9.

Index

A

alphanumeric video output 41
Alt 26
alternate rendering mechanism 37
anti-aliased fonts 16
application extensions 38
application interaction 30
application-defined drag operations 22
application-specified type 27
application-supplied rendering mechanisms and formats 27
application-supplied type string 27
atom table 37, 38
attribute calls 11
augmentation emphasis 25
augmentation key 21
augmentation keys 26

B

bit map calls 12

C

cache presentation space 26, 27
cbDraginfo 19
cbDragItem 19
cditem 19
  DrgSetDragItem 19
child process memory management 43
child windows, help request for 7
clearing the screen 41
client 37
common rendering mechanism and format 24, 31
common type 31
constants 20
contained object 23
container 19
container folder 18
container name 21
container window 17, 23, 25, 26
containers 34, 36
contents of string 27
context information 31
conversation 22, 24, 30, 37
conversation after drop 33
conversation-initiation procedures 32
copy 17, 26, 34, 36, 37, 38
copy operation 31
Courier font 15
cross products 20
cross-product notation 20
Ctrl 26
CUA 25
current mouse-pointer location 19
customized image 27
customized mouse pointer 25, 27

D

data exchange 20, 24
data transfer 32
data-transfer operation 33
database container 21
database manager 21
DDE 20, 30, 36
DDE conversation 37, 38
DDESTRUCT 37
deallocate memory 22
default operation 26
default state 26
define new rendering mechanism 38
delect the string handles 22
delete string handles 35
desktop coordinates 19
device calls 12
directory-navigation 17
DM_DRAGLEAVE 23, 25
DM_DRAGOVER 21, 23, 24
DM_DROP 22, 24
DM_DROPHELP 22, 24
DM_ENDCONVERSATION 34, 35
DM_PRING 36
DM_RENDER 34, 35
DM_RENDERCOMPLETE 34, 35
DM_RENDERPREPARE 32
DOR_DROP 21, 24
DOR_NEVERDROP 21, 24
DOR_NODROP 21, 24
DOR_NODROPOP 21, 24
DOS calls
  DosGiveSeg 43
  DosUnlockSeg 43
DosCopy 38
DosCopy/DosDelete 38
DosGetCollate 59
DosGetCtryInfo 59
DosGetDBCSEv 59
DosMuxSemWait 61
DosPrint API 47
DosPrintDestAdd 48
DosPrintDestControl 48
DosPrintDestDel 48
DosPrintDestEnum 48
DosPrintDestGetInfo 48

DosPrintDestSetInfo 49
DosPrintJobContinue 49
DosPrintJobDel 49
DosPrintJobEnum 49
DosPrintJobGetId 49
DosPrintJobGetInfo 49
DosPrintJobPause 49
DosPrintJobSetInfo 49
DosPrintQAdd 50
DosPrintQContinue 50
DosPrintQDel 50
DosPrintQEnum 50
DosPrintQGetInfo 50
DosPrintQPause 50
DosPrintQPurge 50
DosPrintQSetInfo 50
DosQFileInfo 36
DosRename 38
DosSetFileInfo 36
DO_DEFAULT 19
DO_UNKNOWN 38
DRAFINFO structure 38
DRAFITEM structure 38
drag an object 17
drag string handles 19
drag transfer 35
drag-and-drop 31
dragging objects 21
DRAGIMAGE structure 18
DRAGINFO 22
DRAGINFO structure 18, 23, 24, 26, 32
DRAGITEM structure 20, 26, 30, 32, 35, 36, 37, 38
DRAGTRANSFER structure 35
drawing calls 12
DrgAccessDraginfo 21, 23, 24, 26
DrgAddStrHandle 19, 20, 22, 26, 27
DrgAllocDrag 18
DrgAllocDraginfo 22
DrgAllocDragtransfer 35
DrgDeleteDraginfoStrHandle 22
DrgDeleteDraginfoStrHandles 26
DrgDeleteStrHandle 22, 26
DrgDrag 21, 22
DrgFreeDraginfo 22, 26
DrgFreeDragtransfer 35
DrgGetPS 25, 26
DrgQueryDragitem 26
DrgQueryDragitemCount 26
DrgQueryDragitemPtr 19, 26
DrgQueryNativeRMF 26, 31
DrgQueryNativeRMFLen 26, 31
DrgQueryStrName 27
DrgQueryStrNameLen 27
DrgQueryTrueType 27
DrgQueryTrueTypeLen 27
DrgReleasePS 25, 27
DrgSetDragImage 27
DrgSetDragitem 22
DrgSetDragPointer 25, 27
DrgVerifyNativeRMF 27, 31
DrgVerifyRMF 27, 31
DrgVerifyTrueType 27
DrgVerifyType 27
DrgVerifyTypeSet 27, 31
DrgVerityType 31
drive and path information 21
DRM_DDE 36, 38
DRM_OS2FILE 34
DRM_PRINT 36
dropping the objects 22
DTYP_* constants 19
dynamic data exchange 30

E

end a direct-manipulation operation 22
Esc key 22
establish conversation 32
establishing a conversation 30
exchange data 20, 31
exchange mechanism 30
extended attribute 36, 38
extended rendering mechanism 38

F

file folder 21
file handles (system limits), open 61
file system 45
file trash cans 34
folder 19
font calls 13
fonts 14
fsControl 32
functions used by the source 22
functions used by the target 26

G

GPI MicroPS calls 11

H

handle 37
handle for input string 22
help for the drag operation 24
help requests for a child window 7
Helvetica font 15
highlight phrase table, IPF 63
hot link 37
hrsType field 19
hstrContainerName 34, 35
hstrRenderToName 35, 36
hstrRMF 38
hstrRMF field 30

hstrSourceName 34, 35, 36, 38
hstrType 36, 38
hwndItem 32, 35, 37
hwndSource 32

I

icon 25
icon view 25
Information Presentation Facility
  highlight phrase table 63
  temporary files 9

K

keyboard augmentation 25
keyboard remapping 22

L

length of string 26
limits for number of windows 1
limits, system 61
list box 24
local sharing of segments 43
logical color table calls 13

M

mechanism name 34
memory management
  child process 43
  local sharing 43
  local sharing of segments 43
  process ID 43
message flows 33
messages 21, 34
messages to target application 23
Micro Presentation Space 11
mouse movement 21
mouse pointer 22
move 17, 26, 34, 35, 36, 37, 38

N

name at target 21
name object 18
name view 25
naming convensions 38
naming conventions 32, 35, 38
national language support 59
native copy action 38
native mechanism actions 34
native move action 38
native rendering 34
native rendering by the target 34
native rendering mechanism 35, 36, 37
native rendering mechanism and format 18, 20, 26, 27, 30, 31
no-drop image 22
non-native mechanism 35
notational conveniences 19
null terminating byte 26

O

object element 22
object move 35
object name 38
operation emphasis 26
ordered pair 26, 30
ordered pairs 20, 38
ordered-pair notation 20
OS/2 file 30, 34

P

path name 35
performance considerations 31
pipes, system limits for 61
post-drop conversation 22
preparing for drag 18
Presentation Manager-supplied fonts 15
presentation space 25
Presentation Space calls 13
preventing target rendering 35
print 30, 36
print mechanism 36
print operation 36
printer queue 36
printing an object 36
process ID 43
processes, system limits for 61

Q

queues, system limits 61

R

redefining keys 22
region calls 13
release resources 34
release the storage 22
rendering format 20
rendering mechanism 37
rendering mechanism and format 19
rendering mechanisms and formats 30
rendering operation 34, 35
request a rendering 34
requesting source render 35
responsibility of the target 22, 30
retry 34
return codes, spooler 56

S

screen, clearing the 41
segment sharing 43
semaphores, system limits for 61
server 37
shared memory 22, 35
shared segment 26
shared segments
  child process 43
  DosGiveSeg 43
  local 43
  process ID 43
signals, system limits for 61
single-object move 33
source 17
source application 30
source container 21
source file 35
source name 21, 35
source-specified pointer 22
source-supported formats 37
spin button 3
spooler improvements 47
spooler return codes 56
spreadsheet application 30
standard rendering mechanisms 32, 34
string 26
string handle 19, 27
string name 38
subdirectory name 36
swappable segments 43
Symbol font 15
system limits 61
system-defined rendering mechanisms 32

T

target 17
target application 22, 30
target container 33
target emphasis 24, 25, 26
target object 25
target window 21, 24
text view 25
threads, system limits for 61
Times New Roman font 15
transform 36, 38, 39
transform calls 13
true type 19, 27, 34
two object drag 28
two-object drag 18
type 19
type object 18
type string handle 27
types 36, 38

U

usOperation 19

V

video output, alphanumeric 41
VioSetMode 41
visual cue 25
visual cues 23
visual feedback 22, 24, 25

W

white space 33, 34, 36
window boundaries 23
window handle 22
WinGetKeyState 26
WM_BUTTON2UP 22
WM_DDE_ACK 37
WM_DDE_ADVISE 37
WM_DDE_DATA 37
WM_DDE_INITIATE 37
WM_DDE_REQUEST 36, 37
WM_DDE_TERMINATE 37
WM_DDE_UNADVISE 37
WM_USER 38
writing a target application 23
writing source application 18

X

xDrop 19

Y

yDrop 19

Special Characters

_Format 37
_Operation 26

IBM United Kingdom
International Products Limited
PO Box 41, North Harbour
Portsmouth, P06 3AU
England

Printed in Denmark by Herrmann & Fischer A/S